Parallel Irregular Software for Wave Propagation Simulation

Frédéric Guidec, Patrice Calégari, Pierre Kuonen

Parallel irregular software for wave propagation simulation ⋆

Frédéric Guidec, Patrice Calégari, Pierre Kuonen

Swiss Federal Institute of Technology
Theoretical Computer Science Laboratory
CH-1015 Lausanne, Switzerland
E-mail: {guidec|calegari|kuonen}@di.epfl.ch

Abstract

The objective of the European project STORMS (Software Tools for the Optimization of Resources in Mobile Systems) is to develop a software tool to be used for the design and the planning of the future Universal Mobile Telecommunication System (UMTS). In this context the ParFlow method permits the simulation of outdoor radio wave propagation in urban environment, modeling the physical system in terms of the motion of fictitious microscopic particles over a lattice. This paper gives an overview of the ParFlow method, and reports the design and the implementation of ParFlow++, an object-oriented irregular parallel software for urban outdoor radio wave propagation prediction.

Key words: Mobile telecommunications, radio wave propagation simulation, transmission line matrix, irregular algorithm, object-oriented programming, parallel computation.

1 Introduction

Radio wave propagation simulation is of great interest to telecommunication operators, because of the rapid growth of radio networks, and most especially that of mobile phone networks. This is particularly true in the context of 'cellular networks', for radio wave propagation simulation makes it possible to predict the shape of the cells of any potential future network. The objective of the European project STORMS (Software Tools for the Optimization of

⋆ Expanded version of a talk presented at the High Performance Computing and Networking Europe conference (HPCN, Vienna, Austria, April 1997).

Resources in Mobile Systems) is to develop a software tool to be used for the design and the planning of the future Universal Mobile Telecommunication System (UMTS). This software tool will include radio wave propagation simulation algorithms for both urban and rural environments.

In the STORMS software tool radio wave propagation simulations must be as fast as possible, because a single cellular network may consist of thousands of cells, and because the tool is planned to be used mostly interactively.

The ParFlow method allows for fast bidimensional radio wave propagation simulation, using a digitized city map, assuming infinite building height. It is thus appropriate for simulating radio wave propagation when transmitting antennas are placed below rooftops. This is the case in urban networks composed of micro-cells.

ParFlow++ denotes an object-oriented, irregular implementation of the ParFlow method, targeted at MIMD-DM [1] platforms. Its purpose is to be used in the STORMS project to compute cells covered by Base Transceiver Stations (BTSs) in an urban environment. To date the use of object-oriented programming is not very common in parallel supercomputing. For this reason implementing the ParFlow method using object-oriented techniques appeared to be an appealing challenge.

This paper is organized as follows. Section 2 introduces the ParFlow method, and gives a brief overview of the theoretical foundation of this method (more detailed information can be found in [3,4]). Section 3 reports about the design and implementation of ParFlow++, an object-oriented, parallel implementation of this method. The performances of ParFlow++ are discussed in Section 4, and Section 5 details how the code may evolve in the future. Section 6 concludes with a short summary of the paper.

2 The ParFlow method

In 1995, a new approach to modelling radio wave propagation in urban environments based on a Transmission Line Matrix (TLM [3]) was designed at the University of Geneva by Chopard, Luthi and Wagen [4]. The ParFlow method compares with the so-called Lattice Boltzman Model (LBM), that describes a physical system in terms of motion of fictitious microscopic particles over a lattice [1]. According to the principle of Huygens, a radio 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

[1] Multiple Instruction stream, Multiple Data stream, Distributed Memory.

are represented in terms of finite elementary units Δr and Δt, related by the velocity of light C_0 in such a way that

$$\Delta t = \frac{\Delta r}{C_0\sqrt{2}}$$

Space is modeled by a grid with a mesh size of length Δr, and flow values are defined on the edges connecting neighbouring grid points. The flows entering a grid point at time t are scattered at time $t+\Delta t$ among the four neighbouring points, according to the following expression:

$$\begin{pmatrix} f_1(x+\Delta r, y) \\ f_2(x-\Delta r, y) \\ f_3(x, y+\Delta r) \\ f_4(x, y-\Delta r) \end{pmatrix}_{t+\Delta t} = \frac{1}{2}. \begin{bmatrix} 1 & -1 & 1 & 1 \\ -1 & 1 & 1 & 1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & -1 & 1 \end{bmatrix} . \begin{pmatrix} f_1(x, y) \\ f_2(x, y) \\ f_3(x, y) \\ f_4(x, y) \end{pmatrix}_t$$

The computation proceeds until a stable state is reached, or until a predefined number of iteration steps has been run through.

Because of the discretization of time, it is convenient to distinguish between the flows coming into a grid point, and those going out of this point. For each grid point, there are thus four incoming flows, and four outgoing flows, that all obey the following rules:

- outgoing flows at time t are a linear combination of incoming flows at that time;
- an incoming flow at time t corresponds to the outgoing flow calculated on a neighbouring grid point at time $t - \Delta t$;
- the field value of a grid point at time t is the sum of the four flows coming into this point at time t.

These rules apply for all grid points but those modelling the source (the transmitter) and obstacles. The source point does not propagate incoming flows. Instead, by changing its excitation function one can simulate different kinds of output signals, such as a sinusoidal signal, a triangular pulse, an impulse, etc. Obstacles (typically, city buildings) are modelled by two kinds of grid points: wall points, and indoor points (see Figure 1). As, in the current ParFlow model, it is assumed that radio waves do not penetrate buildings, indoor points are not involved in the computation. Wall points are perfectly reflecting points that return any incident radio wave with opposite sign, and whose reflection coefficient and matrix elements can be adapted in order to model different kinds of walls [6]. An additional kind of point is considered in the ParFlow method. Since the method must simulate the (theoretically) infinite

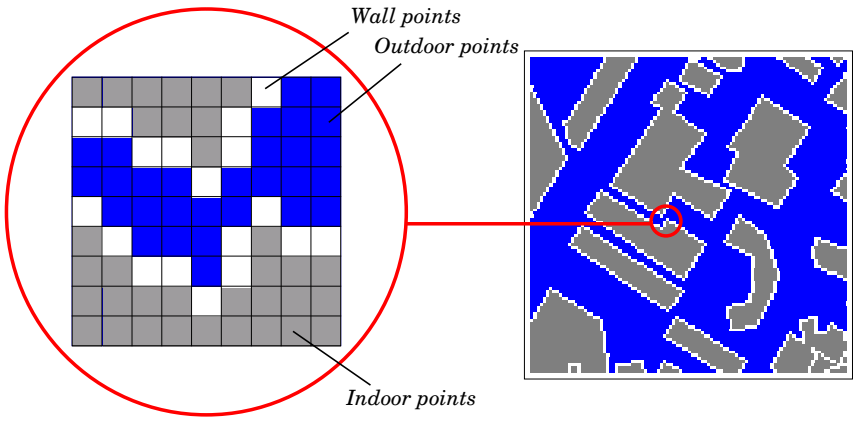


Fig. 1. The ParFlow method operates on a city district described as a bitmap that distinguishes between indoor points, outdoor points, and wall points.

propagation of a radio wave on a finite grid, special points are placed around the simulation zone in order to absorb part of the incident radio wave. These border points simulate the attenuation of radio waves propagating in open air.

3 Design and implementation

3.1 Object-oriented irregular application

The grid structure on which the ParFlow algorithm operates, and the way each point of the grid is updated iteratively based on its four neighbours, suggest a very regular implementation. A language primarily dedicated to array-based computation, such as Fortran, is perfectly suited for this kind of implementation. Yet, since the ParFlow method does not simulate radio wave propagation through buildings, it can be most interesting not to model indoor points. Indeed, experiments show that when modelling an urban area, buildings can represent up to 30 % of the surface considered. Not modelling indoor points avoids a waste of memory space and a waste of computational power. 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 appear as good candidates for an irregular implementation of the ParFlow method.

ParFlow++ was developed in C++, in an object-oriented way, and according to the fundamental principles of software engineering. All kinds of non-indoor points (that is, outdoor points, wall points, border points, and the source point) are described in a hierarchy of C++ classes. Instances of these classes can be assembled at runtime so as to constitute an irregular structure that preserves the neighbourhood relationships: each point references any of its four

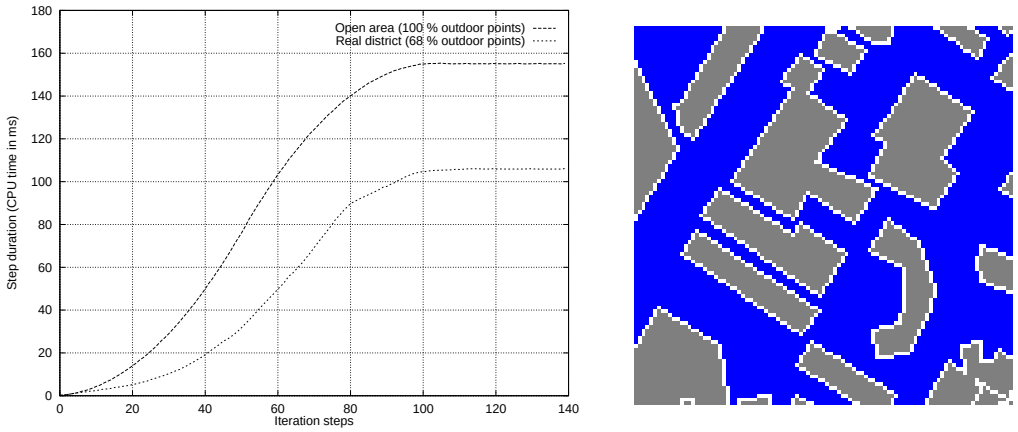
neighbours (provided they exist).

In ParFlow the amount of computation required during a simulation step is not constant. Because of the discretization of time and space in the simulation process, a radio wave radiated by the source point propagates step by step throughout the simulation zone. Based on these observations, ParFlow++ was designed in such a way that the amount of points implied during any computation step is always kept at a minimum. At any time, a point that has not been reached by the radio wave yet is said to be inactive (initially, all points are inactive but the source point). Points that have just been reached by the radio wave (during the current simulation step) are said to be *new* active points, whereas those that were reached during former simulation steps are referred to as *old* active points.

At each step of the simulation flow values need only be calculated for active points, and the propagation of outgoing flows is only required from active points to their neighbours. To achieve these goals ParFlow++ manages several structures internally. Every newly activated point is inserted in a list *newActive*, whose role is to permit an efficient propagation of the activity status after each computation step: the only points that are in a position to be activated are those that are neighbours to members of the *newActive* list, and that are still inactive. Iterating through members of the *newActive* list facilitates the identification of new active points. Once a member of *newActive* has activated its neighbours, it is transferred into another list *oldActive*. Hence, this point will never be considered again when looking for new active points, although it will still participate in the computation. A sequential version of the algorithm operating on active lists is reproduced below.

```
newActive = {source point};
oldActive = ∅;
foreach iteration step do
    foreach point ∈ oldActive do
        compute outgoing flows based on incoming flows;
        update incoming flows based on neighbours' outgoing flows;
    endfor
    oldActive = oldActive ∪ newActive;
    tmpActive = newActive;
    newActive = ∅ ;
    foreach point ∈ tmpActive do
        compute outgoing flows based on incoming flows;
        update incoming flows based on neighbours' outgoing flows;
        activate not-yet-activated neighbours;
        newActive = newActive ∪ {newly activated neighbours};
    endfor
endfor
```

With a regular, grid-based implementation of the ParFlow method, there is no



(a) (b)

Fig. 2. Evolution of the workload per simulation step during a sequential simulation on a 100×100 points area. The results in (a) show the workload evolution observed when achieving a simulation on an open area (upper curve), and on a real district of the city of Geneva (lower curve). The district actually considered in the latter simulation is reproduced in (b). The CPU times reported in (a) were measured on one processor of a Cray T3D supercomputer.

notion of *active* points. All grid points participate in the computation, so that the workload per simulation step remains constant all along the computation. With ParFlow++ the workload per simulation step increases dynamically, as shown in Figure 2. This figure confirms the advantage of activating points dynamically. It shows how the workload per simulation step increases as the radio wave propagates and covers a growing surface, and how it reaches a ceiling when the simulation zone is completely covered.

Figure 2 also confirms the advantage of using a data structure that does not model indoor points. The speed at which the workload increases depends on the amount of obstacles met by the radio wave during its propagation, and the maximal workload depends on the amount of outdoor points in the simulation zone. In Figure 2(a) it is interesting to notice that the gain observed in terms of workload reduction is proportional to the amount of indoor surface.

3.2 Data parallel implementation

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 algorithm was implemented by Chopard,

Luthi and Wagen on Thinking Machine Corporation's CM-200, whose SIMD [2] architecture provides thousands of synchronous processors [6]. 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 presented in Section 3.1 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 modelling indoor points in the case of ParFlow) 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 [5]. 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 mostly 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. To do so, its current implementation uses the communication facilities offered by the standard PVM library [2].

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 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.

In ParFlow++ a partition basically consists of a structured collection of points and of two lists that are used to maintain references to 'new' and 'old' active points (these are the two lists discussed in Section 3.1). Each partition 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 behaviour of frontier points differs slightly from that of other points, for they have to interact with neighbouring points that are stored

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

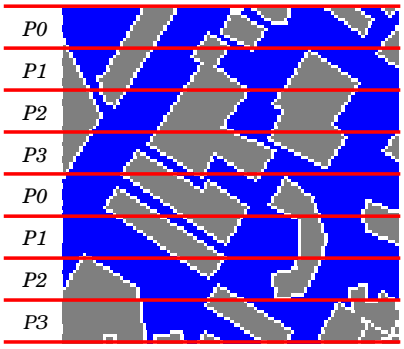


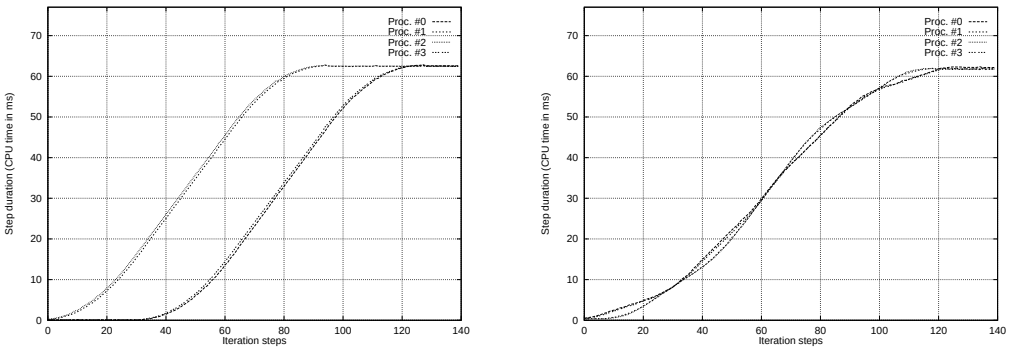
Fig. 3. Splitting of the simulation zone in 8 horizontal stripes, and allocation of these stripes to 4 processors using a round-robin policy.

on remote processors. A frontier thus ensures the propagation of flows between a partition and one of its neighbours. It also ensures the propagation of the activity status, since a newly active point located on a frontier may activate a point in the opposite frontier of a neighbouring partition. The parallel version of the ParFlow algorithm is reproduced in pseudo-code below.

```
newActive = {source point};
oldActive = ∅;
foreach iteration step do
    foreach local partition do
        // Same loop body as that shown in Section 3.1
        send northern and southern frontiers to neighbouring partitions;
        receive northern and southern frontiers from neighbouring partitions;
        newActive = newActive ∪ {new active points found in frontiers};
    endfor
endfor
```

Partitions dimensioning. Partitioning the simulation zone in stripes has several advantages. Mechanisms for data location and message vectorization are readily implemented. Moreover, as adjacent stripes can be allocated to adjacent processors, communications are only required between neighbouring processors.

When partitioning the simulation zone in stripes, the width of these stripes and the number of them assigned to each processor can be adjusted easily. It is thus possible to obtain a fair distribution of the workload among the processors, as confirmed by the results reproduced in Figure 4. These results were observed when achieving a parallel radio wave propagation simulation on a 128×128 points open area (no obstacle, source point located in the middle of the simulation zone), on 4 processors of the Cray T3D MC 256. Figure 4 shows how the workload per processor depends on the partitioning policy. The results reproduced in Figure 4(a) were observed with a simulation zone split in 4 partitions (one partition per processor). The workload increased rapidly on



(a) (b)

Fig. 4. Evolution of the workload during a parallel radio wave propagation simulation achieved on 4 processors of a Cray T3D, for a 128×128 points open area (no obstacle). (a) and (b) show how the workload evolves when the simulation zone is split in 4 partitions (one partition per processor), and in 16 partitions (4 partitions per processor) respectively.

processors 1 and 2, because the points in these partitions were activated early during the simulation process. On the other hand, processors 0 and 3 started working only after the 31^{st} iteration step. This is because it took 32 iteration steps for the radio wave to reach the partitions owned by processors 0 and 3. Figure 4(b) shows the workload evolution when the simulation zone was split in 16 partitions (4 partitions per processor). This time the workload increased almost similarly on all processors. This is because of the large number of partitions, and because these partitions were thin: all processors got their share of work early after the radio wave started propagating.

These results show that partitioning the simulation zone in thin stripes permits a better balancing of the workload. Yet, this approach also leads to a greater communication overhead, for each partition must exchange data with its two neigbouring partitions after each simulation step.

4 Experimental results

On the Cray T3D we run a series of simulations for different partitioning policies, and different simulation zone sizes. Figure 5 shows the path-loss map obtained after simulating an 800 step propagation on a 500×500 point zone modelling a 1 km^2 district of the city of Geneva. This simulation took 1109 seconds on a single PE (Processing Element) of the Cray T3D, and 42 seconds on 32 PEs.

We also studied the influence of data partitioning on the resulting perfor-

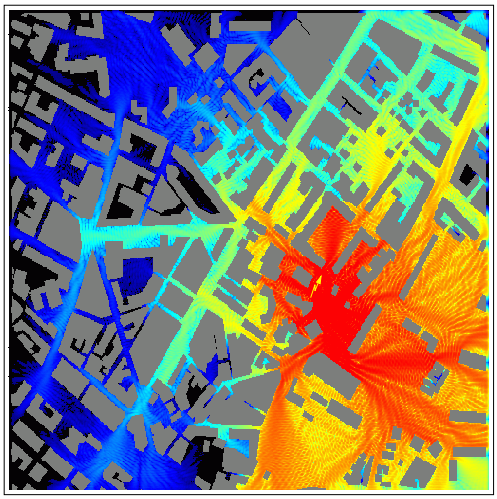
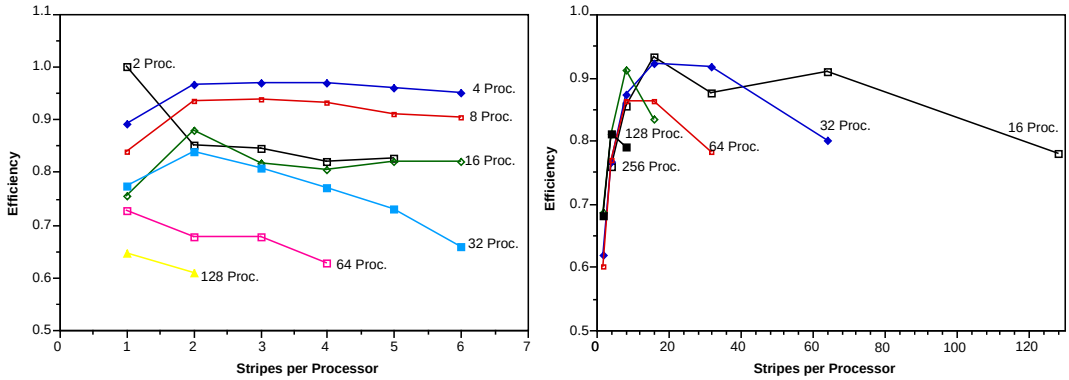


Fig. 5. Results of a radio wave propagation simulation achieved for a 1 km^2 district of the city of Geneva. The broken arrow shows the position of the transmitter.



(a) (b)

Fig. 6. Efficiency of parallel simulations on a Cray T3D as a function of the number of stripes per processor. (a) shows the results observed with a 512 × 512 point simulation zone, and (b) those observed with a 2048 × 2048 point zone.

mances. During these experiments we considered wave propagation over open areas (no obstacle), and in each test case the source point was placed in the middle of the simulation zone. Figure 6 shows how the number of stripes per processor influences the global performances of the simulation. The figure shows the efficiency observed, for different hardware configurations (from 2 to 256 processors), as a function of the number of stripes per processor, for 512 × 512 point and 2048 × 2048 point simulation zones.

Figure 6(a) shows that, for each machine size, there usually exists an optimal partitioning policy. On the one hand, when using only two processors, there is no point in having more than one stripe per processor. Since, in this particular simulation case, the source of the wave is located in the middle of the

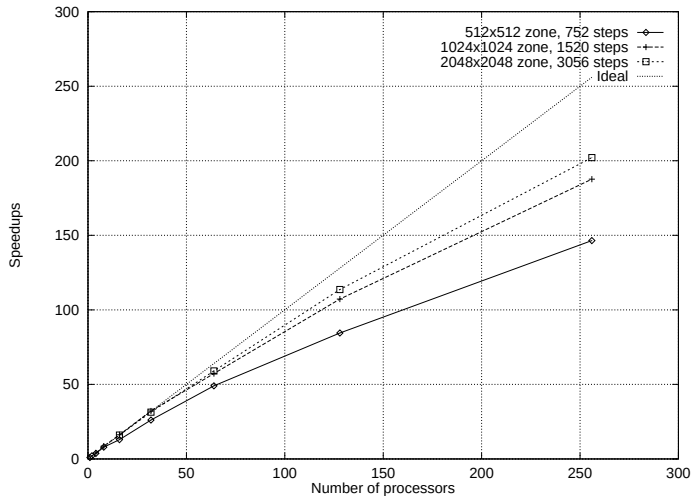


Fig. 7. Speedups observed on a Cray T3D, when achieving simulations for various district sizes.

simulation zone, both processors get exactly the same share of work. Partitioning the zone in more than two partitions results in a greater communication overhead, hence a lower efficiency. On the other hand, when using more than two processors it can be interesting to have several stripes per processor (although 2 or 3 stripes usually give the best efficiency). This is not true any more, though, when using more than 32 processors. This is because, when partitioning a 512×512 point zone in more than 32 processors, the height of each stripe then gets so small that the ratio of the computation time over the communication time is not good enough.

The advantage of allocating several stripes per processor can also be observed with the 2048×2048 point simulation zone (Figure 6(b)), except that it can then be interesting to allocate up to 8 stripes per processor.

Figure 7 shows the *best* speedups we observed (that is, whatever the number of partitions) for different zone sizes [3], against the number of processors. These results confirm the scalability of the parallel implementation.

5 Future work

The performances reported in the former section are quite satisfactory. Yet, ParFlow++ can still be improved in many ways. For example the code shown in Section 3 could be modified so that after a certain number of simulation

[3] For the 1024×1024 and 2048×2048 zones, the simulation was not possible on a single processor because of memory limitation, so we had to estimate the sequential reference times.

steps the search for still inactive points is disabled (once the radio wave has covered the entire simulation zone, there remains no such points).

The results reported in Section 4 show that partitions dimensioning is a crucial issue. Some work is now in progress to build a mathematical model of communications and workload in ParFlow++ [7]. This model should help select the best partitioning policy for any simulation problem case.

We would also like to experiment with a heterogeneous partitioning policy. Since many stripes are required near the source point in order to give better workload balancing, and since too many stripes lead to a degradation of the communication performances, a compromise would be to allocate thinner stripes near the source.

Once ParFlow++ has been improved satisfactorily, it will be interesting to study how its performances compare with those of regular, grid-based implementations of the ParFlow method, such as those discussed in [5]. ParFlow++ will also be tested more comprehensively on the Cray T3D –using up to 256 processors– as well as on other MIMD-DM platforms (so far ParFlow++ was compiled and tested successfully on a Cray T3D, on a SGI Origin 2000, and on a cluster of Unix workstations).

6 Conclusion

In this paper we have reported the development of ParFlow++, a new parallel object-oriented implementation of the ParFlow method. ParFlow++ permits the prediction of radio wave propagation in urban environments, based on a bidimensional simulation model over a digital city map.

The most original feature of ParFlow++ with respect to former implementations of the ParFlow method is that it was implemented as an irregular, distributed, application. Since the method does not allow for radio waves to propagate through buildings, ParFlow++ does not model indoor points. Experiments show that this approach permits a significant reduction of computation times.

ParFlow++ was designed in an object-oriented framework, and implemented in C++ for MIMD-DM platforms. It was developed so as to be highly portable, and its current implementation relies on the PVM library. Although this implementation can still be improved in many ways (as discussed in Section 5), experiments achieved on a Cray T3D show the scalability of the code and confirm that an object-oriented irregular implementation for MIMD-DM platforms does not necessarily lead to poor performances.

In contrast, the object-orientation of ParFlow++ brings in much versatility. The set of classes that constitute its source code could easily be extended or reused in another context. For example ParFlow++ could be modified so as to simulate tri-dimensional radio wave propagation. Its classes could also serve as basic building blocks to develop software tools capable of simulating other physical phenomena, such as fluid dynamics or reaction-diffusion processes.

Acknowledgement

The STORMS project is a European ACTS project, funded by the European Community and by the Swiss government (OFES grant). The Cray T3D experimentations were conducted on the machine of the Swiss Federal Institute of Technology in Lausanne.

References

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

[2] A. Geist and al. *PVM: Parallel Virtual Machine. A User's Guide and Tutorial for Networked Parallel Computing.* The MIT Press, 1994.

[3] W. J. R. Hoeffer. The Transmission-Line Matrix Method: Theory and Applications. *IEEE Transactions on Microwave Theory and Techniques*, 33(100):882–893, oct 1985.

[4] P. O. Luthi and B. Chopard. Wave Propagation with Transmission Line Matrix. Technical report, University of Geneva and Swiss Telecom PTT, 1994.

[5] P. O. Luthi, B. Chopard, and J.-F. Wagen. Radio Wave Propagation Simulator on Scalable Parallel Computers. In N. Droux, editor, *Proceedings of the SIPAR'95 Workshop, Parallel and Distributed Systems*, pages 63–66. Biel School of Engineering, CS Department, CH-2501 Biel-Bienne, Switzerland, Oct. 1995.

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

[7] M. Pahud, F. Guidec, and T. Cornu. Performance Analysis of a Parallel Program for Wave Propagation Simulation. In S. Lengauer, M. Griebl, and S. Gorlatch, editors, *Euro-Par'97 Parallel Processing (Third International Euro-Par Conference, Passau, Germany, August 1997, Proceedings)*, volume 1300 of *Lecture Notes in Computer Science*. Springer, Sept. 1997.

Frédéric GUIDEC was born in Loudéac, France, on March 23, 1966. He received his Ph.D. in Computer Science from the University of Rennes, France, in 1995. Since then he has been a member of the Parallel Computing Research Group at the Swiss Federal Institute of Technology (Lausanne, Switzerland). His research interests include parallel computing, software engineering, as well as object-oriented technology applied to telecommunications and supercomputing.

Patrice CALÉGARI was born in Saint Julien-en-Genevois, France, on November 9, 1969. He received the M.S. degree in Computer Science from the University of Lyon, France, in 1994. He is now pursuing a Ph.D. in Computer Science at the Swiss Federal Institute of Technology (Lausanne, Switzerland). His research interests are focused on parallel computing, evolutionary algorithms and combinatorial optimization.

Pierre KUONEN was born in Switzerland on July 23, 1958. He obtained a degree in Electrical Engineering from the Swiss Federal Institute of Technology (EPFL) in 1982. After six years of experience in industry he joined the Computer Science Theory Laboratory at the EPFL in 1988. He then started working in the field of parallel computing, and received his Ph.D. degree in 1993. Since 1994 he has been a scientific collaborator, heading the Parallel Computing Research Group of the Computer Science department. He is a lecturer in parallel computation courses.