HAL
open science

Object-Oriented Parallel Software for Parallel Radio Wave Propagation Simulation in Urban Environment

Frédéric Guidec, Patrice Calégari, Pierre Kuonen, Michel Pahud

To cite this version:

Frédéric Guidec, Patrice Calégari, Pierre Kuonen, Michel Pahud. Object-Oriented Parallel Software for Parallel Radio Wave Propagation Simulation in Urban Environment. Computers and Articicial Intelligence, 1999, 18 (6), pp.525-539. hal-00346558

HAL Id: hal-00346558

https://hal.science/hal-00346558v1

Submitted on 11 Dec 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Computers and Artificial Intelligence, Vol. 18, 1999, No. 6, 1—

OBJECT-ORIENTED PARALLEL SOFTWARE FOR RADIO WAVE PROPAGATION SIMULATION IN URBAN ENVIRONMENT

Frédéric Guidec
Patrice Calégari
Pierre Kuonen
Michel Pahud

Swiss Federal Institute of Technology
Theoretical Computer Science Laboratory
Parallel Computing Research Group
CH-1015 Lausanne, Switzerland
E-mail: {guidec,calegari,kuonen,pahud}@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, the implementation, and the performance analysis of ParFlow++, an object-oriented irregular parallel software for urban outdoor radio wave propagation prediction.

Keywords: Radio wave propagation simulation, Transmission Line Matrix, mobile telecommunication, parallel programming, irregular algorithm, object-oriented programming, performance modeling, Bulk Synchronous Programming.

1 INTRODUCTION

Because of the rapid growth of cellular phone networks, radio wave propagation simulation is of great interest to telecommunication operators, for it makes it possible to predict the shape of the cells of any potential future networks. The objective

of the 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). This software tool, which is planned to be used mostly interactively, shall include radio wave propagation simulation algorithms for both urban and rural environments. Since a single cellular network may consist of thousands of cells, radio wave propagation simulation must be as fast as possible.

The ParFlow method permits the simulation of outdoor radio wave propagation in urban environment, describing the physical system in terms of the motion of fictitious microscopic particles over a lattice. It is appropriate for simulating radio wave propagation when fixed antennas are placed below rooftops, as is the case in urban networks composed of micro-cells.

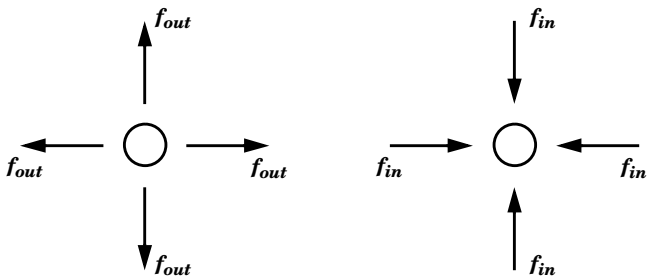
This paper reports the design and the implementation of ParFlow++, an object-oriented, irregular implementation of the ParFlow method targeted at distributed memory parallel platforms. To date the use of object-oriented programming is not very common in parallel supercomputing. This is the reason why the parallel implementation of the ParFlow method using object-oriented techniques appeared to us as 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 [6, 8]). Section 3 reports about the design and the implementation of ParFlow++, an object-oriented, parallel implementation of this method. The performances of ParFlow++ are discussed in Section 4, and Section 5 shows how these performances can be predicted accurately for any problem case and any target parallel platform. Section 6 concludes with a short summary of the paper.

2 THE PARFLOW METHOD

The ParFlow method was designed at the University of Geneva by Chopard, Luthi and Wagen [8]. It compares with the so-called Lattice Boltzman Model, that describes a physical system in terms of motion of fictitious microscopic particles over a lattice [1]. 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*. 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 (Fig. 1). The flows entering a grid point at time t are scattered at time $t + \Delta t$ among the four neighbouring points.

* $\Delta t = \dfrac{\Delta r}{C_0\sqrt{2}}$.



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

Fig. 1. Each grid point interacts with its neighbours in the grid through four outgoing flows and four incoming flows.

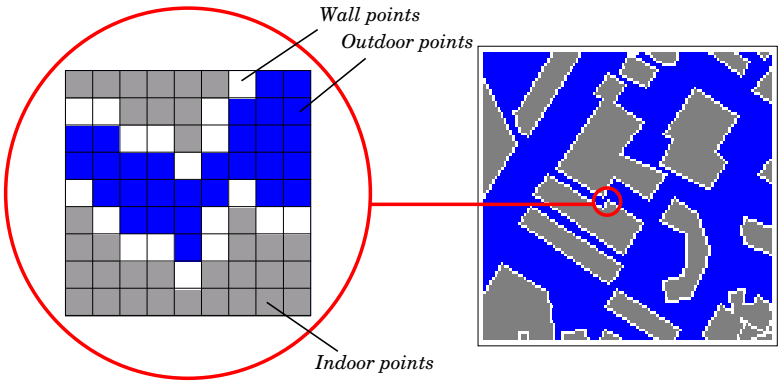


Fig. 2. The ParFlow method operates on a city district described as a bitmap that distinguishes between indoor points, outdoor points, and wall points.

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. 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 neighbouring grid point at time $t - \Delta t$. These rules apply for all grid points but those modeling the source (the transmitter) and obstacles. The source point radiates a signal through its four outgoing flows, but it does not propagate incoming flows. Obstacles (typically, city buildings) are modeled by two kinds of grid points: wall points, and indoor points (see Fig. 2). In the current ParFlow model it is assumed that radio waves do not penetrate buildings, indoor points are thus 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 [9].

3 DESIGN AND IMPLEMENTATION

3.1 Object-oriented irregular application

Since the ParFlow method does not simulate radio wave propagation through buildings, it can be most interesting not to model indoor points. Indeed, experience shows that when modeling an urban area, buildings can represent up to 30 % of the surface considered. Not modeling 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 are perfectly suited to an irregular implementation of the ParFlow method.

ParFlow++ was thus developed in C++, according to the fundamental principles of software engineering. The preliminary analysis and design of the application were achieved using the Fusion method [2]. The main classes and the relationships that were identified during the analysis phase constitute an object model, which is partially reproduced in Fig. 3.

All kinds of non-indoor points 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.

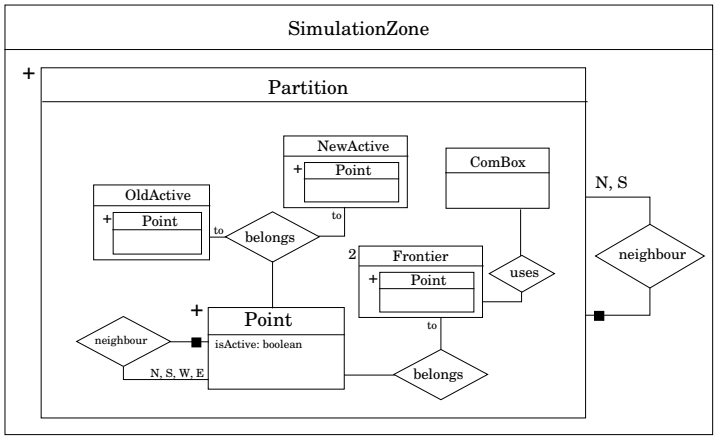
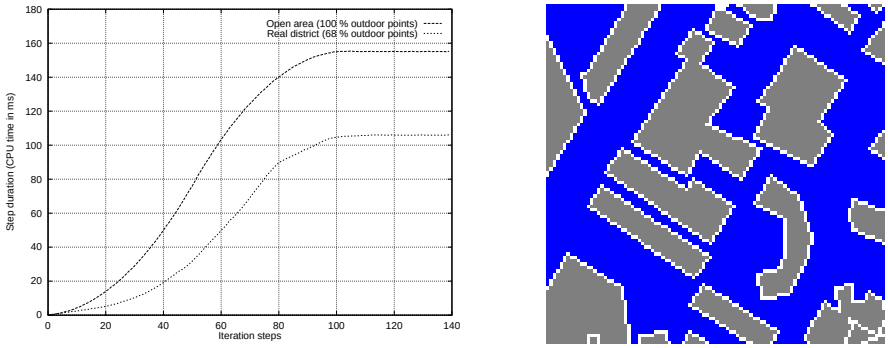


Fig. 3. Fusion object model of ParFlow++.



(a) (b)

Fig. 4. Evolution of the workload per simulation step during a sequential simulation on a 100 × 100 point 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.

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 wave radiated by the source point propagates step by step throughout the simulation zone. 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 wave yet is said to be inactive (initially, all points are inactive but the source point). Points that have just been reached by the 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.

Fig. 4 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.

Fig. 4 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.

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.

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 Fig. 5.

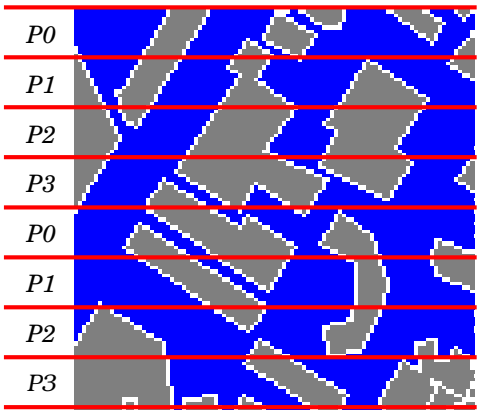


Fig. 5. Split of the simulation zone in 8 horizontal stripes, and allocation of these stripes to 4 processors using a round-robin policy.

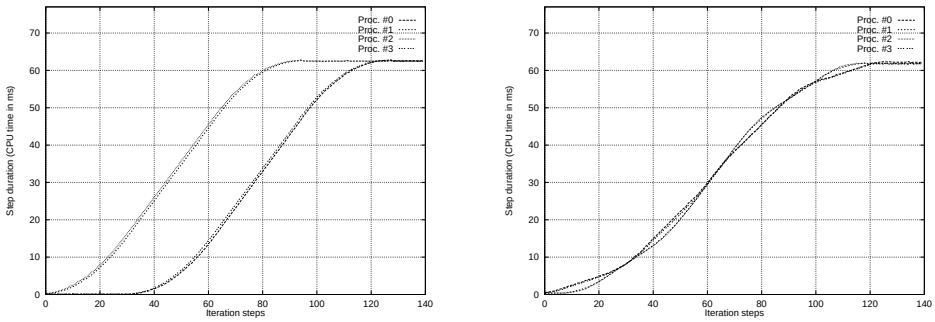
In ParFlow++ a partition basically consists of a structured collection of points and of two lists that are used to maintain references to 'new' and 'old' active points (these are the two lists discussed in Section 3.1). Each partition also manages two 'frontiers' internally. A frontier can be perceived 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 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.

ParFlow++ 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 [4]. The use of another library (MPI, for exemple) would only require the change of a few C++ classes dedicated to the communications. The overall application weights more than 15000 lines of C++ code.

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 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 Fig. 6. These results were observed when achieving a parallel radio wave propagation simulation on a 128×128 point open area (no obstacles, source point located in the center of the simulation zone), on 4 processors of the Cray T3D MC 256. Fig. 6 shows how the workload per processor depends on the partitioning policy. The results reproduced in Fig. 6a were observed with a simulation zone split in 4 partitions (one partition per processor), and those reproduced in Fig. 6b were obtained with a simulation zone split in 16 partitions (4 partitions per processor).

These results confirm 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. The latter observation is investigated further in the next section, and Section 5 shows how a performance prediction model makes it possible to choose the best partitioning policy for any simulation problem case.



(a) (b)

Fig. 6. Evolution of the workload during a parallel radio wave propagation simulation achieved on 4 processors of a Cray T3D, for a 128 × 128 point open area (no obstacles). (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.

4 EXPERIMENTAL RESULTS

On the Cray T3D we ran a series of simulations for different partitioning policies, and different simulation zone sizes. Fig. 7 shows the path-loss map obtained after simulating an 800 step propagation on a 500 × 500 point zone modeling 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 (hence a speedup of 26.40, and an efficiency of 83 %).

We also studied the influence of data partitioning on the resulting performances. During these experiments we considered wave propagation over open areas (no obstacle), and in each test case the source point was placed in the center of the simulation zone.

Fig. 8 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.

Fig. 8a 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 center of the 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 distributing a 512 × 512 point zone over more

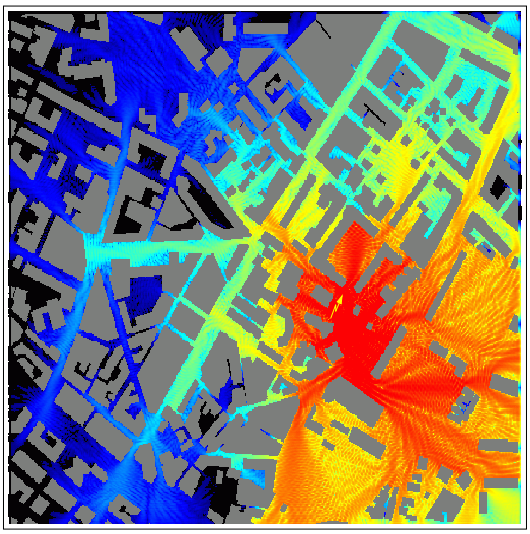
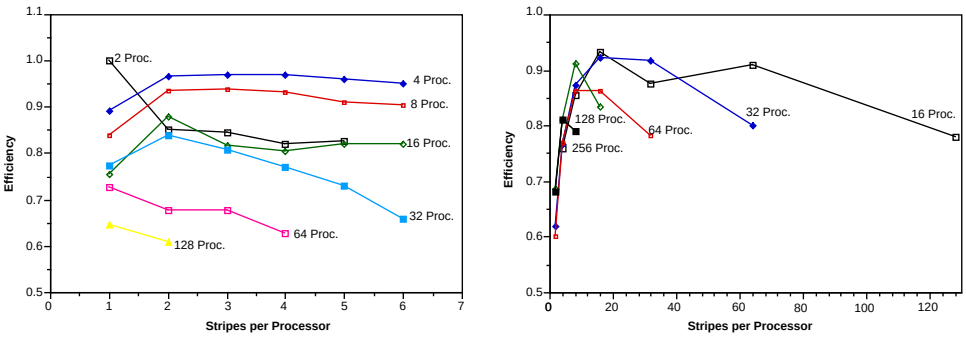


Fig. 7. Results of a radio wave propagation simulation achieved for a 1 km^2 district of the city of Geneva. The broken arrow shows the position of the transceiver.



(a) (b)

Fig. 8. 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.

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 (Fig. 8b), except that it can then be interesting to allocate up to 8 stripes per processor.

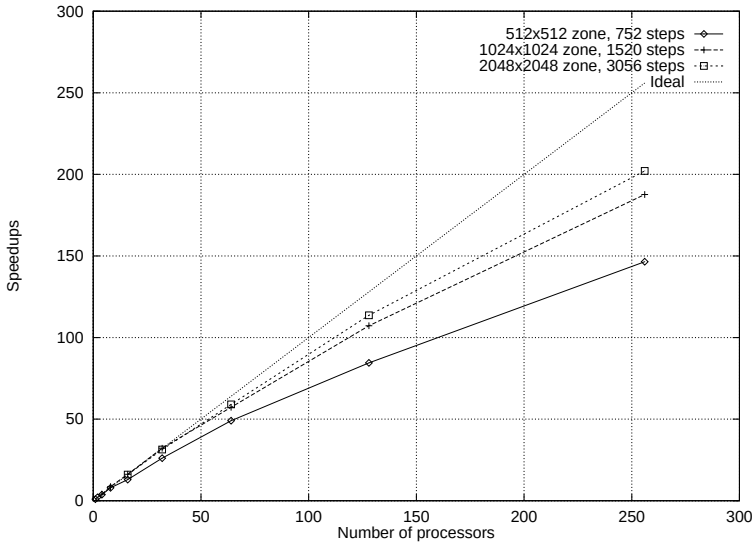


Fig. 9. Speedups observed on a Cray T3D, when achieving simulations for various district sizes.

Fig. 9 shows the *best* speedups we observed (that is, whatever the number of partitions) for different zone sizes**, against the number of processors. These results confirm the scalability of the parallel implementation, but they also show that partitions dimensioning is a crucial issue for getting the best performances out of ParFlow++. This is the reason why a mathematical model of the workload and of communications in ParFlow++ was designed so as to help select the best partitioning policy for any simulation problem case. This model is described in the next section.

5 PERFORMANCE MODELING

During the last decade, performance prediction was repeatedly quoted as a key factor to developing parallel systems [3, 5, 7]. Predicting the performance of a parallel program such as ParFlow++, as a function of the number of processors and of the problem size, is crucial for choosing the best hardware configuration and for tuning various parameters.

The model we developed for ParFlow++ is closely related to Valiant's well-known *Bulk Synchronous Programming* (BSP) model [10]. It relies on the mea-

** 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.

surement of basic communication and computation routines. The computational workload of each processor and the load imbalance are modeled analytically.

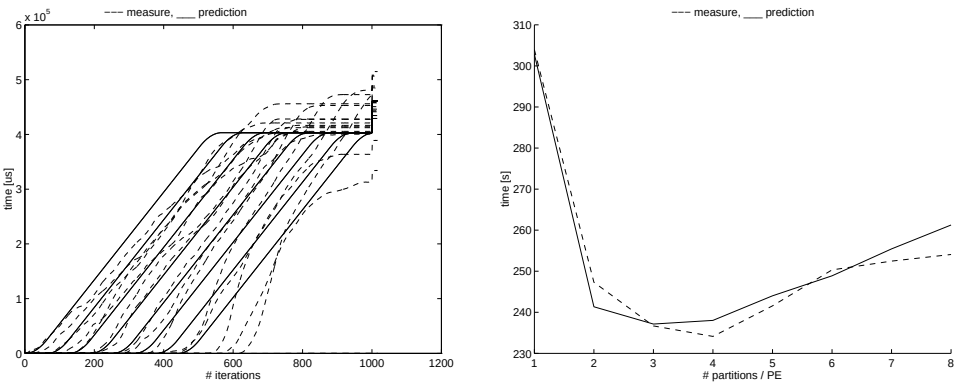
Our performance model focuses on the behaviour of the ParFlow++ algorithm while performing the actual wave propagation simulation. The computation time of one iteration step is assumed to be that needed by the most heavily loaded processor, and the computation workload of a processor is assumed to be proportional to the number of active points handled by this processor. This number is not necessarily identical for each processor and increases at each iteration step. The cost of the communication itself is assumed to be proportional to the number of partition borders handled by one processor. This cost is constant over the iterations since the volume of data transferred only depends on the number of points per partition border.

To model the behaviour of ParFlow++ on a new parallel architecture, only two measurements are required: the computation time for each active point, and the communication time per partition border. The computation time per active point is obtained by timing one iteration of a sequential ParFlow++ execution, when all grid points are active. The communication time is obtained by timing the point-to-point communication time of a message of the same size as a partition border. Contention is neglected in a first approximation. The measured communication time includes the cumulated time taken by the communication routine, by the procedures responsible for packing and unpacking the data contained in a border, and finally by the call to the encapsulating C++ methods.

To model local computation, we evaluate the number of active points handled by a processor at a given time as a function of the number of processors and of the number of partitions per processor. The number of active points is first predicted for a void area (i.e., without buildings). When buildings are present the actual number of active points is obtained by subtracting all indoor points, since they are not processed in the ParFlow++ implementation. To avoid considering the actual location of buildings on the grid, we assume that they are uniformly distributed.

Results. The ParFlow++ performance model was validated on the Cray T3D multi-processor architecture. Fig. 10a compares the measured and the theoretical computation times for a 16-processors system working on a 1000×1000 simulation zone (namely, a district of the city of Geneva), with one partition per processor. Communication times are not considered here. Each curve shows the computation time spent by one of the processors as a function of the iteration step i.

Fig. 10a confirms that the model fits quite well to the actual results. Discrepancies between them show the impact of the non uniform distribution of buildings. Although the load of individual processors is predicted only imperfectly, this prediction can still be used very efficiently in the estimation of the total execution time. Fig. 10b shows the predicted and measured execution times for the district of the Geneva city, as a function of the number of partitions per processor. The



(a) (b)

Fig. 10. Workload distribution (a) and execution times (b) of ParFlow++ running on 16 processors. Dashed curves show actual measurements and plain curves show predicted values.

communication time is taken into account. Both the predictions and the measurements clearly show the trade-off between a good load-balancing and reduced communication costs. The optimal partitioning for this problem size is actually four partitions per processor, while the prediction indicates three partitions. The performance difference between the predicted and measured option is quite small. The discrepancies between the two curves are probably due to the irregularity of the distribution of buildings. Without *a priori* information on building locations, three partitions per processors would actually be the best bet.

The evolution of the measured and predicted speedups for the same area was also compared, as a function of the number of processors used. In this case, only one partition per processor was used. The quality of the prediction was very satisfactory: prediction error varied between 0 and 6%.

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, 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.

Experimentation shows that performance prediction is a crucial issue with parallel, irregular algorithms such as that of ParFlow++. The performance model we designed for ParFlow++ is based on the BSP model [10]. It requires only a few basic measurements: the timing of point-to-point communications and of elementary sequential computations. This model has proved very accurate when predicting performances of runs on the T3D. In the future it can thus be used for scalability analysis (such as speedup prediction), or to seek optimal trade-offs in issues such as data partitioning.

Acknowledgements. The STORMS project is a European ACTS project, funded by the European Community and by the Swiss government (OFES grant). The work on performance modeling was partly supported by the Swiss National Science Foundation under grant # 2100-042391.94/1. The Cray T3D experimentations were conducted on the supercomputer of the Swiss Federal Institute of Technology in Lausanne.

REFERENCES

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

[2] Coleman, D. et al.: Object-Oriented Development – The Fusion Method. Prentice Hall Object-Oriented Series, Englewood Cliffs, NJ, 1995. ISBN0-13-338823-9.

[3] Fahringer, T.: Automatic Performance Prediction for Parallel Programs on Massively Parallel Computers. PhD thesis, Technical University Vienna, Austria, 1993.

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

[5] Gupta, A.—Kumar, V.: Analyzing scalability of parallel algorithms and architectures. Journal of Parallel and Distributed Computing, 22(3):379–391, 1994.

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

[7] Kuhn, W.: Performance prediction and benchmarking results from the ALPSTONE project. In: Proceedings of the International Conference and Exhibition on High-Performance Computing and Networking (HPCN Europe'96), No. 1067 in Lecture

Notes in Computer Science, pages 763–769, Brussels, Belgium, Apr. Springer Verlag 1996, pp. 763–769.

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

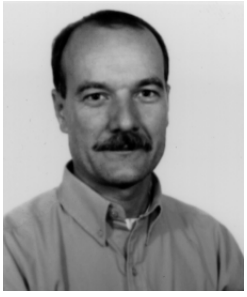
[9] Luthi, P. O.—Chopard, B.—Wagen, J.-F.: Wave Propagation in Urban Microcells: a Massively Parallel Approach using the TLM Method. In: Proceeding of PARA '95, Workshop on Applied Parallel Scientific Computing, Copenhagen, August 1995. Also in COST 231 TD(95) 33.

[10] Valiant, L. G.: Bulk-synchronous parallel computers. In: M. Reeve and S. E. Zenith (Eds.): Parallel Processing and Artificial Intelligence, Chichester, UK, Wiley 1989.

Manuscript received 30 January 1998; revised 8 October 1998

Frédéric Guidec was born in France in 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 France in 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 in 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.

Michel Pahud was born in Switzerland 1968. He received his engineering diploma in computer science from the Swiss Federal Institute of Technology (EPFL) in 1995. He is now pursuing a Ph.D., in the domain of performance analysis of parallel irregular applications. The goal of this work is to provide models of architecture-application pairs, suitable to make performance predictions of parallel computation. He also works as an assistant and lecturer in the EPFL's Microprocessors Systems laboratory.