Performance Evaluation of a Radio Wave Propagation Parallel Simulator

Michel Pahud, Frédéric Guidec, Thierry Cornu

Performance Evaluation of a Radio Wave Propagation Parallel Simulator

Michel Pahud, Frédéric Guidec, Thierry Cornu
Swiss Federal Institute of Technology Lausanne
Parallel Computing Research Group
Theoretical Computing Laboratory
CH-1015 Lausanne, Switzerland
E-mail: {pahud,guidec,cornu}@di.epfl.ch

Abstract

This paper presents a performance model for a parallel application with irregular data structures, that simulates wave propagation. The model is based on the measurement of basic communication and computation routines. The computational workload of each processor and the load imbalance are modeled analytically. Comparison between predicted and measured performance are shown for a Cray T3D computer.We indicate how the model can be used to analyze the scalability of the algorithm and to find optimal trade-offs in the partitioning of its data.

1 Introduction

During the last decade, performance prediction has been repeatedly quoted as a key factor to developing parallel systems [3, 4, 6]. Predicting the performance of a program as a function of the number of processors and of the problem size is important, for choosing the best hardware size and and tuning various parameters. Numerous prediction tools have been recently proposed in the literature [2, 7, 8, 9, 13, 14]. Most of them focus mainly on regular applications using static data structures. Some approaches exclusively consider loop nests [4].

In contrast, this paper shows an example of performance analysis of an irregular application. The application is a wave propagation simulation algorithm called ParFlow++. The performance analysis method is closely related to the *Bulk Synchronous Programming* (BSP) model [12, 15]. The wave propagation program is described in section 2. The performance prediction is based on the measurements of communication times on the parallel machine used (a Cray T3D), and on the execution times of sections of the code on one processor. Section 3 describes the performance model, section 4 compares the estimated performance with actual measurements, and section 5 concludes the paper.

2 Wave propagation with ParFlow++

2.1 ParFlow method

The ParFlow method was designed at the University of Geneva by Chopard, Luthi and Wagen [10]. It is based on the so-called Lattice Boltzman Model, describes a physical system in terms of motion of fictitious microscopic particles over a lattice [1]. In Parflow, a radio-electric wave is simulated by the propagation of a flow inside of a two-dimensional discrete grid, according to a finite difference equation model.

Obstacles (typically, city buildings) are modeled by two kinds of grid points: wall points, and indoor points (see figure 1). In the current ParFlow model, it is assumed that waves do not penetrate buildings: wall points are perfectly reflecting points that return any incident wave with opposite sign [11]. Indoor points may therefore be ignored in the computation.

2.2 ParFlow++ implementation

Since the ParFlow method does not simulate wave propagation through buildings, it can be most interesting to avoid any computation for 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 wasting memory space and saves computational power. An other improvement is also possible: 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; no

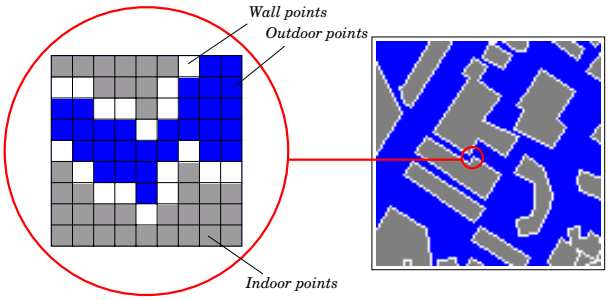


Figure 1. The ParFlow method operates on a city district described as a bitmap that distinguishes between indoor points, outdoor points, and wall points.

```
activePoints = {source point};
foreach iteration step do
    tmpActive = activePoints;
    foreach point in tmpActive do
        compute outgoing flows based on incoming flows;
        update incoming flows based on neighbors' outgoing flows ;
        activePoints = activePoints
                        ∪ {newly activated neighbors};
    endfor
endfor
```

Figure 2. Single-processor version of the ParFlow++ algorithm.

computation should be performed for points not yet reached by the wave.

ParFlow++ [5] is a C++ implementation of the ParFlow method that implements these optimizations. ParFlow++ maintains a list of references to *active points*. Active points are grid points already reached by the wave and not located inside buildings.

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 neighbors. A single-processor version of the algorithm is reproduced in figure 2.

Due to the presence of buildings at random locations in the grid, the ParFlow++ program maintains irregular data structures. The amount of computation required during a simulation step also varies in an irregular way.

2.3 Multi-processor implementation

Like other grid-based computations, the ParFlow method is a good candidate for parallel implementation. A parallel version of the ParFlow++ code was developed for MIMD machines with a distributed memory. It relies on the PVM communication library. The simulation zone is split into

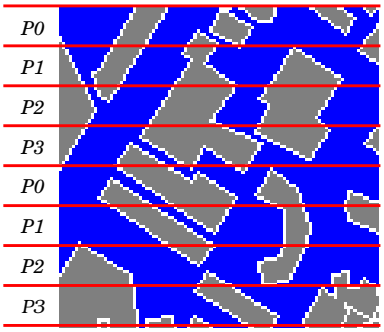


Figure 3. Simulation zone partitioning and mapping.

```
activePoints = {source point};
distribution of partitions onto the processors;
forall processors do in parallel
    foreach iteration step do
        localTmpActive = localActivePoints;
        foreach point in localTmpActive do
            compute outgoing flows based on incoming flows;
            update incoming flows based on neighbors' outgoing
flows;
            localActivePoints = localActivePoints
                                ∪ {newly activated neighbors};
        end foreach
        communication of partition borders;
        localActivePoints = localActivePoints
                            ∪ {newly activated border points};
    end foreach
end forall
```

Figure 4. Multi-processor version of the ParFlow++ algorithm.

horizontal strip called partitions, which are then allocated to processors based on a round-robin policy, as shown in figure 3.

A local list of active points is maintained for each partition. Due to the partitioning policy, adjacent partitions are always allocated to different processors. Thus, communications take place at the borders between adjacent partitions, were wave flows and activation status are propagated.

The multi-processor implementation of ParFlow++ is depicted in figure 4.

During the initial stage of the wave propagation, only the processors whose partitions are close to the wave source host active points. This results in a load imbalance. Balancing of the workload is achieved by increasing the number of partitions allocated per processor, as shown in figure 5. On the other hand, multiplying the number of partitions increases the communication costs. Therefore, a trade-off has to be found in the number of partitions per processor. Predicting the optimal number of partitions is one of the goals

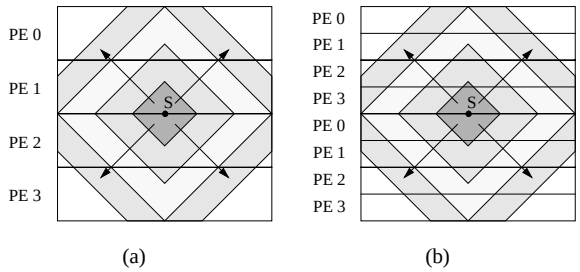


Figure 5. Simulation of wave propagation on four processors: (a) one partition per processor, (b) two partitions per processor.

of the performance model developed in the next section.

3 Performance model

Our performance model focuses on the outer **foreach** loop of the algorithm of figure 4. The initialization phase of the algorithm, that is, the distribution of initial data onto the processors, is not considered in this paper. In fact, modeling the initialization phase is also not relevant for determining the optimal number of partitions.

3.1 Methodology

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. A model of the number of active points per processor, and of its evolution, is detailed in section 3.2.

Since the communication operation at the end of the **foreach** loop implements an implicit synchronization barrier, the computation time of one iteration is assumed to be the one needed by the most heavily loaded processor.

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, in our implementation, the volume of data transferred does not depend on the number of active points but only on the number of points per partition border.

To model the behavior of ParFlow++ on a new parallel architecture, only two measurements will be required: the execution time of one iteration of the innermost loop, and the communication time per partition border. The execution time of the innermost loop is measured by timing one iteration of a single-processor ParFlow++ algorithm, when all grid-points are active (an area without buildings is considered). 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 PVM 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.

3.2 Cost of local computation

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, of the number of partitions per processor, and of the position of the considered processor's partitions with respect to the source of the wave. We first derive the number of active points for an area *without buildings*. When the wave source is located in the middle of the simulation area, deriving the number of points reached by the wave is straightforward. The following notations will be used in this paper:

- i - the current iteration step;
- d - the iteration step at which the considered partition is entered by the wave; assuming no buildings, d depends only on the distance of the wave source to the nearest edge of the partition;
- L - the partition length in grid points;
- h - the height of the partition in grid points;
- $w(i)$ - the number of active points in the partition at iteration step i; this quantity is proportional to the computational workload induced by the partition.

In the case of $L \geq 2h$, the propagation of the wave inside a partition can be subdivided into three stages (figure 6):

1. the first stage starts when the wave enters the partition through one of its borders, and ends when the opposite border is reached;
2. the second stage ends when the wave reaches the left and right edges of the partition;
3. the last stage ends when all points of the partition have been reached.

Depending on the maximal number of iterations, stages 2 and 3 may be completed only partially, or not at all. Once all partition points have been activated, computation may still proceed further. However, the number of active points of the partition has reached a maximum and becomes a constant value. The number of grid points reached by the wave at the different stages can be computed:

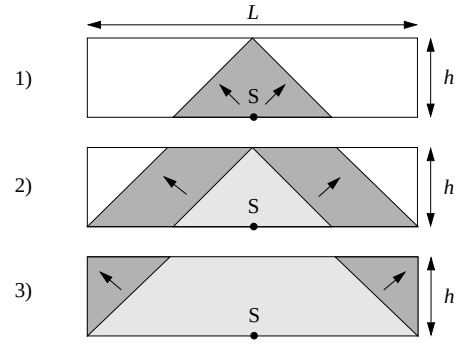


Figure 6. Three successive stages of the propagation of the wave inside a partition.

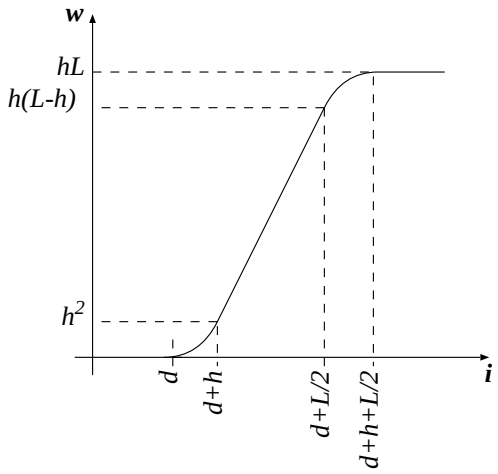


Figure 7. Number of grid points reached by the wave as a function of the iteration.

$$
\begin{array}{ll}
d \le i \le d+h & \rightarrow w(i) = (i-d)^2 \\
d+h < i \le L/2 & \rightarrow w(i) = 2h(i-d-h)+h^2 \\
d+L/2 < i \le d+h+L/2 & \rightarrow w(i) = hL-(h+L/2-i+d)^2 \\
i > d+h+L/2 & \rightarrow w(i) = hL
\end{array}
$$

This yields the curve of figure 7.

When buildings are present, the number of active points is derived from the previous model, by subtracting all indoor points, since they are not processed in the ParFlow++ implementation. To avoid considering the actual location of buildings in the grid, we assume that they are uniformly distributed. Therefore, the number of active points is simply multiplied by a constant corrective factor, corresponding to the ratio of outdoor surface to the total surface.

In the case of $L < 2h$, or for a wave source not centered, the number of stages to distinguish in the propagation changes. In these cases, it is straightforward to generalize the above analysis, although the complexity of the calculation will increase.

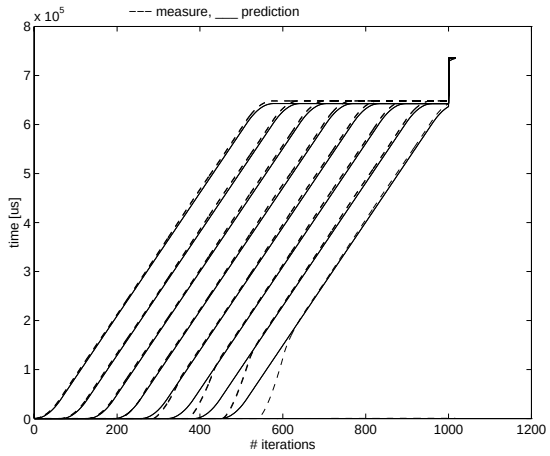


Figure 8. Time spent by the different processors in the main ParFlow++ loop (measured and predicted) as a function of the iteration step. Data for a 1000 $\times$ 1000 simulation zone without buildings, simulated on 16 processors with 1 partition each. Dashed curves correspond to actual measurements and plain curves to the model.

4 Results

We validated the ParFlow++ performance model on the Cray T3D multi-processor architecture. Let us first examine the local computational workload as described in section 3.2. Figure 8 compares the measured and the theoretical computation times for a 16-processors system working on a 1000×1000 simulation zone *without buildings*, with one partition per processor. Communication times are not considered here; each curve shows the computation time spent by one of the processors in the body of the outermost **foreach** loop of figure 4, as a function of the iteration step i. In this case, the model fits well to the data. The hypothesis that the computation time is proportional to the number of active points is verified.

Figure 9 shows similar results, but with buildings present (namely, a district of the city of Geneva). Since no computation is performed in the indoor areas, the average execution time at the end of the propagation (iteration step 1000) is lower than for the empty area. Discrepancies between the measurements and the model show the impact of the non uniform distribution of buildings.

Although, the load of individual processors is predicted only imperfectly (due to the irregularity of data), this prediction can still be used very efficiently in the model of the total

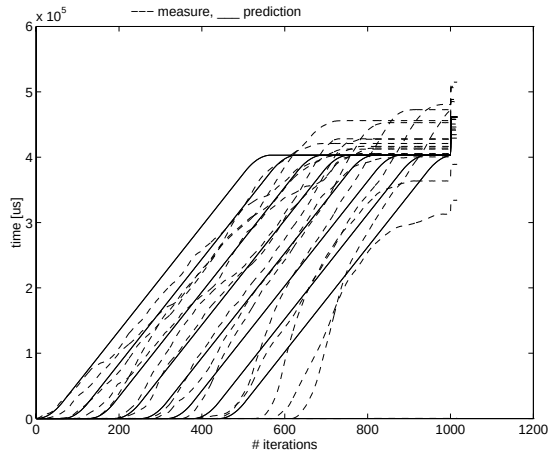


Figure 9. Time spent by the different processors in the main ParFlow++ loop (measured and predicted) as a function of the iteration step. Data for a 1000 $\times$ 1000 simulation zone with buildings, representing a 2 km $\times$ 2 km district of Geneva, simulated on 16 processors with 1 partition each. Dashed curves correspond to actual measurements and plain curves to the model.

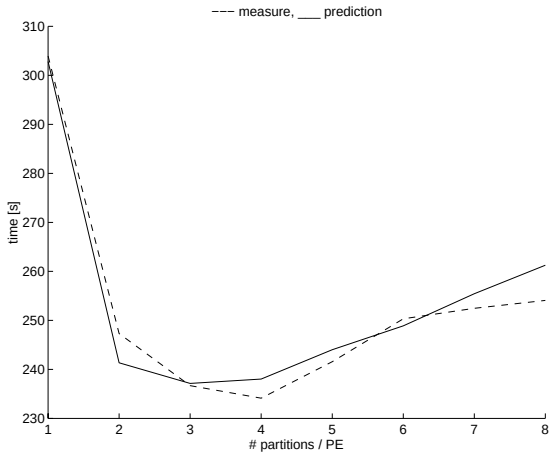
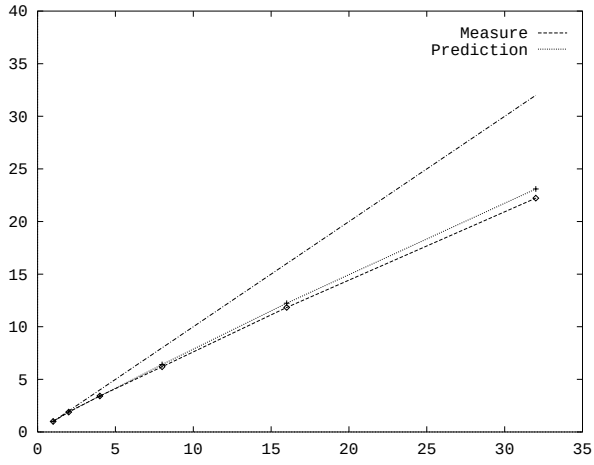


Figure 10. Total execution time of ParFlow++ on a 2 km $\times$ 2 km district of the Geneva city (1000 $\times$ 1000 points), as a function of the number of partitions per processor.

execution time. This is shown on figures 10 and 11. Figure 10 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 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 mainly 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.

Figure 11 compares the evolution of the measured and predicted speedups for the same area, as a function of the number of processors used. In this case, only one partition per processor is used. The quality of the prediction is very satisfactory; prediction error varies between 0 and 6%.



5 Conclusion

We have presented a performance model of a parallel application using an irregular data structure. On the massively parallel Cray T3D, we were able to provide a valid predic-

tion. The model presented can be used for scalability analyses, such as speedup prediction, or to seek optimal tradeoffs in issues such as data partitioning. Prediction relies on only a few basic measurements: the timing of point-to-point communications and of elementary local computations.

Future work will include the modeling of other parallel irregular algorithm with a similar methodology. Parallel optimization algorithms like branch & bound, tabou search and evolutionary algorithms, are good candidates for such investigations. Modeling on other multi-processor architectures (other distributed memory machines like the Intel Paragon, shared memory architectures like the SGI Origin 2000, and networks of workstations) will also be investigated, to check the generality of the modeling method.

The approach we adopted will be applied to other applications and systematized into a general performance model. The BSP model [12] will give a very good framework for this purpose. BSP has been shown to be very well adapted to the performance modeling of a lot of parallel regular algorithms, and we expect it to also fit to a wide range of irregular algorithms.

Acknowledgments

This research was supported partly by the Swiss National Science Foundation under grant # 2100-042391.94/1. The ParFlow++ application was developed in the framework of the European ACTS project STORMS. The Cray T3D experimentations were conducted on the machine of the Swiss Federal Institute of Technology in Lausanne.

The following people deserve acknowledgments for helpful discussions and proofreading the manuscript: G. Coray, C. Diderich, P. Kuonen.

References

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

[2] R. J. Block, S. Sarukkai, and P. Mehra. Automated performance prediction of message-passing parallel programs. In *Proceedings of Supercomputing'95*, San Diego, Dec. 1995. ACM/IEEE.

[3] N. J. Davies. *The Performance and Scalability of Parallel Systems*. PhD thesis, Department of Computer Science, Faculty of Engineering, University of Bristol, UK, Dec. 1994.

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

[5] F. Guidec, P. Kuonen, and P. Calégari. ParFlow++: A C++ parallel application for wave propagation simulation. *SPEEDUP Journal*, 10(1/2):68–73, Dec. 1996.

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

[7] K. Harzallah and K. C. Sevcik. Predicting application behavior in large scale shared-memory multiprocessors. In *Proceedings of Supercomputing'95*, San Diego, Dec. 1995. ACM/IEEE.

[8] J. M. D. Hill, I. Crumpton, Paul, and A. Burgess, David. Theory, practice, and a tool for BSP performance prediction. In *Proceedings of the Euro-Par'96 Conference, volume II*, number 1124 in Lecture Notes in Computer Science, pages 697–713, Lyon, Aug. 1996. Springer Verlag.

[9] W. Kuhn. 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)*, number 1067 in Lecture Notes in Computer Science, pages 763–769, Brussels, Belgium, Apr. 1996. Springer Verlag.

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

[11] P. O. Luthi, B. Chopard, and J.-F. Wagen. Wave Propagation in Urban Micro-cells: 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.

[12] W. F. McColl. Universal computing. In *Proceedings of the Euro-Par'96 Conference, volume I*, number 1123 in Lecture Notes in Computer Science, pages 25–36, Lyon, Aug. 1996. Springer Verlag.

[13] R. Sarukkai, Sekhar, P. Mehra, and R. J. Block. Automated sclability analysis of message-passing parallel programs. *IEEE Parallel and Distributed Technology*, Winter 1995.

[14] J. Simon and J.-M. Wierum. Accurate performance prediction for massively parallel systems and its applications. In *Proceedings of the Euro-Par'96 Conference, volume II*, number 1124 in Lecture Notes in Computer Science, pages 675–688, Lyon, Aug. 1996. Springer Verlag.

[15] L. G. Valiant. Bulk-synchronous parallel computers. In M. Reeve and S. E. Zenith, editors, *Parallel Processing and Artificial Intelligence*, Chichester, UK, 1989. Wiley.