Performance Analysis of a Parallel Program for Wave Propagation Simulation

Michel Pahud, Frédéric Guidec, Thierry Cornu

Performance Analysis of a Parallel Program for Wave Propagation Simulation

Michel Pahud, Frédéric Guidec, Thierry Cornu

Swiss Federal Institute of Technology Lausanne
Parallel Computing Research Group
EPFL-DI-LITH
CH-1015 Lausanne
E-mail: [pahud,guidec,cornu]@di.epfl.ch

1 Introduction

During the last decade, performance prediction has been repeatedly quoted as a key factor to developing parallel systems [1, 2, 3]. Predicting the performance of a parallel program 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.

This paper presents a method for achieving performance analysis for parallel irregular applications. The model is closely related to the *Bulk Synchronous Programming* (BSP) model [4]. It is based on the measurement of basic communication and computation routines. The computational workload of each processor and the load imbalance are modeled analytically.

The method is used for predicting the performances of ParFlow++, an irregular, parallel radio-wave propagation algorithm.

2 Wave propagation with ParFlow++

The ParFlow simulation method is based on the so-called Lattice Boltzman Model. It describes a physical system in terms of motion of fictitious, microscopic particles over a lattice. 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. Indoor points may therefore be ignored in the computation.

ParFlow++[i] is a data-parallel C++ implementation of the ParFlow method that implements these optimizations. It maintains a list of references to *active points*, that is, outdoor grid points that have already been reached by the wave. At each step of the simulation, values need only be calculated for active points, and the propagation of outgoing flows is only required from active points to their neighbors.

[i] For a full description of ParFlow++ see the paper *'Object-Oriented Parallel Software for Radio Wave Propagation Simulation in Urban Environment'*, also published in these proceedings.

The simulation zone is split into horizontal strips called partitions, which are then allocated to processors based on a round-robin policy. Communications take place at the borders between adjacent partitions, where wave flows and activation status are propagated. Balancing of the workload is achieved by increasing the number of partitions allocated per processor. Yet, increasing the number of partitions increases the communication costs. Therefore, a trade-off must be found in the number of partitions per processor. Predicting the optimal number of partitions is one of the goals of the performance model developed in the next section.

3 Performance model

Our performance model focuses on the behavior of the algorithm while performing the actual wave propagation simulation. The computation time of one iteration 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 behavior 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 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.

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.

4 Results

The ParFlow++ performance model was validated on the Cray T3D multiprocessor architecture. Figure 1(a) compares the measured and the theoretical computation times for a 16-processors system working on a 1000 $\times$ 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.

Figure 1(a) confirms that the model fits quite well to the actual results. Discrepancies between them show the impact of the non uniform distribution of buildings.

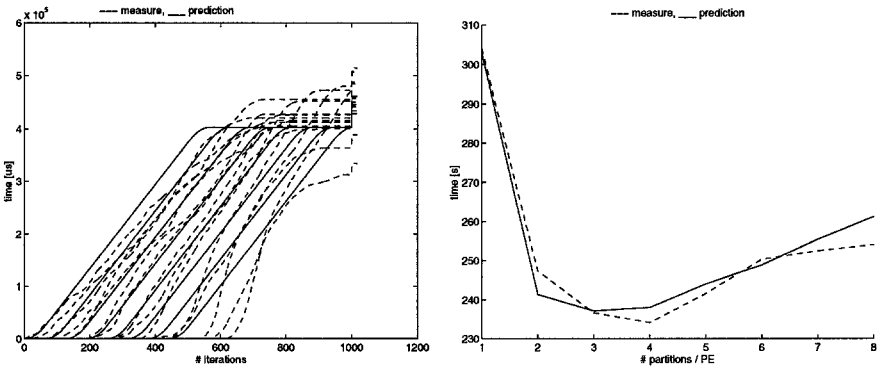

Fig. 1. Workload distribution (a) and execution times (b) of ParFlow++ running on 16 processors. Dashed curves show actual measurements and plain curves show predicted values.

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. Figure 1(b) 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 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%.

5 Conclusion

The performance model for a parallel, irregular application presented is based on the BSP model [4], known for its ability to represent regular algorithms. It was tested on an irregular simulation problem and was able to provide a valid prediction for executions on the Cray T3D. It can be used for scalability analyses, such as speedup prediction, or to seek optimal trade-offs in issues such as data partitioning. Prediction relies on only a few basic measurements: the timing of point-to-point communications and of elementary sequential 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. On other multi-processor architectures (distributed memory machines such as the Intel Paragon, shared memory architectures like the SGI Origin 2000, and networks of workstations) performance prediction will also be investigated to check the generality of the modeling method. In the long term, our objective is the development of a generic prediction tool, that should process annotated parallel irregular programs automatically and predict their performances on any target parallel platform.

References

1. T. Fahringer. *Automatic Performance Prediction for Parallel Programs on Massively Parallel Computers.* PhD thesis, Technical University Vienna, Austria, 1993.
2. A. Gupta and V. Kumar. Analyzing scalability of parallel algorithms and architectures. *Journal of Parallel and Distributed Computing,* 22(3):379–391, Sept. 1994.
3. 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.
4. L. G. Valiant. Bulk-synchronous parallel computers. In M. Reeve and S. E. Zenith, editors, *Parallel Processing and Artificial Intelligence,* Chichester, UK, 1989. Wiley.