Perspectives in Performance Evaluation of Large ATM Networks

C.D. Pham S. Fdida

Laboratoire LIP6
Université Pierre et Marie Curie
4 place Jussieu 75252 Paris Cedex 05
e-mail : {pham,fdida}@masi.ibp.fr
phone: (33-1) 44-27-75-12 fax: (33-1) 44-27-62-86

Abstract

The demand for broadband communication is becoming more and more urging. In order to achieve a successful expand of the ATM technology, performance evaluation of large networks is a mandatory step and the challenge is now to simulate them in one piece. Distributed simulation is a promising tool for the simulation of such systems that can not be handled sequentially anymore. This paper presents our experiments in distributed simulation of a large ATM network at the cell-level on parallel machines. The models include a real routing algorithm, connection setup and VPI/VCI translation. First, a network of 26 switches is simulated on a CM-5 and on a Cray T3E. The results show that a maximum speedup of 4.5 can be obtained. Second, a larger network that contains 78 switches is simulated on the Cray T3E. This experiment shows that a speedup of 7.68 can easily be obtained.
Keywords: Distributed simulation, ATM networks.

1 Introduction

The performance evaluation field consists in the analysis and the design of systems. These systems can be whatever we want as long as performance criterion can be extracted for analysis. Two main methods are used to perform this task: the analytical approach and the discrete simulation approach. Analytical equations provide an elegant way to model and obtain stable results. However, these approaches often require simplifications that reduce the usefulness of the study. Simulation is more likely to allow a complete study since the level of detail is not theoretically limited. Therefore, it is not surprising that simulation has gain a lot of interest over these past years to become a widespread real performance evaluation tool. In addition to be flexible, many people can use it with only a short training period whereas a strong background in mathematics is required for analytical methods.

Telecommunication networks is a perfect example of a large and complex system. As the demand for broadband communication is becoming larger and larger it becomes very urging to offers such high-speed backbones that would provide sufficient bandwidth for a various number of applications such as multimedia, video-conferencing and TV distribution. Although simulations can be carried for small or specific parts of the network (i.e. switch element for switch architecture design), the challenge is now to simulate them in one piece. In an environment where both speed and bandwidth are increased, problems like congestion avoidance, traffic control and optimal routing algorithms should be addressed first to ensure the best behavior of the network. It is not only necessary to evaluate new algorithms but also the existing that eventually perform well in actual networks but may fail in such an environment.

Each solution, analytical methods or simulation, has its own disadvantages. The flexibility of simulation is paid by a large amount of computation time and it is clear that the size and the complexity of nowadays systems is such that sequential simulation has reach its limits. Over the last decade, there has been a growing interest in bringing the power available in

parallel machines to the simulation task. These techniques are classified as either *conservative* or *optimistic* depending on the underlying synchronization method.

Distributed simulation techniques are very promising. However, due to the lack of flexible and powerful tools these techniques are still unknown to a large part of discrete-event simulation practitioners. The work we conducted leads us to develop a kernel based on the distributed conservative approach along with various ATM models. This paper presents our experiments in distributed simulation of large topologies at the cell-level on parallel machines. Performing at the cell-level gives the possibility to inspect the smallest interactions between network components. Previous works by Earnshaw and Hind [4], Mouftah and Sturgeon [11] to cite only two, have also investigated the communication network distributed simulation problem but they are limited to the simulation of simple transfer of packet/cell between nodes with uniform routing. The speedup reported in [4] is almost linear for the most favorable case when compared to an optimized version of the parallel simulator. Our work includes the routing algorithm described in [8] with connection management. The simulation of a routing algorithm implies to simulate (*i*) the mechanism that consists in constructing and updating (on a cost change such as a link failure) the routing table and (*ii*) the flow of cells transported by the network. The distributed simulations are performed on a Connection Machine (CM-5) of the *Centre National de Calcul Parallèle en Science de la Terre* (CNCPST) and a Cray T3E of the *Institut du Développement et des Ressources en Informatique Scientifique* (IDRIS).

This paper is organized as follows: the next section presents some backgrounds in distributed simulation. Section 3 presents the ATM network models and the routing algorithm. Section 4 describes briefly our conservative simulation kernel and presents the speedup measures. Conclusion and future works are then given in Section 5.

2 Distributed simulation background

Simulation can either be *synchronous* or *asynchronous*. Time-step approaches belong to the first category and present 2 main disavantages for us. First, a synchronous simulator would constraint the time unit to the smallest time increment that exists in the whole system. When a large variety of time increment exists (i.e. transmission rates are not homogeneous) this approach would lead to an overwhelming overhead. Second, a time-step approach is difficult to distribute because the definition of a global clock is hard. Therefore we focus mainly on the *discrete event* approach where changes in the system occur at discrete points in the simulated time with the model jumping from one point to another according to the progress of event processing. The distributed approach usually assumes that the system to be simulated can be spatially partitioned into disjoint sub-systems. Each sub-system is simulated by an associated *Logical Process* (LP) on a dedicated processor. To model the interactions that may exist in the real system messages are exchanged and timestamped by the sending LP according to its *Local Virtual Time* (LVT). At the receiving side, they are put in a local *Future Event List* (FEL) to be processed. Other methods to parallelize a simulation exist but their applications are limited [5, 6].

Care must be taken when distributing virtual clocks and event lists to ensure that the distributed simulation is still *correct* and produce the same output when compared to a sequential one. In particular, the *causality constraints* that represent the logical evolution of time must be maintained. Unfortunately since the different LPs may advance at different rates synchronization problems are likely to occur resulting in a *causality error* when a message arrives at a receiving LP and is outdated, or old, according to its LVT. These problems arise the need of explicit synchronization mechanisms that are traditionally classified in two categories: *conservative* and *optimistic*.

The conservative approach [2] only processes events that can definitely be considered safe, i.e. that processing the event would not result in further causality violations. Safety is guaranteed by forcing the LP to block when bad decisions can be taken. The advantage is to obtain a correct simulation at any time but this scheme introduces deadlocks that must be avoided by sending

null-messages. These messages rely on *lookahead* ability to artificially propagate and advance the simulation time. An alternative that consists in detecting the deadlock and breaking it also exists. On the other side, optimistic approaches [9] do not search for safety but provisions are made to *roll back* to an earlier coherent state when they occur. Periodic check-pointing and *anti-messages* to cancel bad computations are then needed as a counterpart of more freedom. In addition, a *Global Virtual Time* (GVT) is required to monitor the simulation progress and to reclaim memory used by obsolete information.

Experimental studies have shown that neither conservative nor optimistic approaches can be used for a large variety of applications. In this work, we choose a conservative approach as the underlying synchronization method for our kernel. This choice is dictated by two main reasons. First, some thoughts suggest that communication network models are not well-suited for optimistic scheduling since they have a very low computation granularity that would propagate bad computations very quickly. And second, a conservative method is much more simple than an optimistic one and therefore is more likely to be efficiently implemented by a small team.

3 Modeling the ATM network

Communication networks provide facilities for the transfer of information between end-points of the network. In the ATM technology, data are sent through the network in fixed-size packets, namely cells, by means of ATM switches linked together by optical fiber links. At the cell-level only two main events should be considered: a reception event, noted $< recv, t_r >$, that indicates at the receiving LP a cell reception event at time t_r and a send event[1], noted $< send, t_d >$, that indicates at the sending LP the departure of a cell at time t_d.

Modeling the network requires to model the switch fabric and the traffic of cells at the inputs. Traffic modeling is a major effort and we do not intend to address this wide problem at the moment. It is simply assumed that a traffic source generates $< recv, t_r >$ events where t_r is distributed according to a simple statistical law (exponential, constant...). We now describe the switch model and the network topology for the study.

3.1 The switch model

ATM switches are the core of the transfer technology and a lot of architectures have been proposed in the literature [3, 7]. From a strictly simulation protocol point of view, details of the switch architecture do not really matter. It is always possible to assume that (i) a switch continually receives $< recv, t_r >$ messages in its FEL, each message generating a sequence of n events $< e_1, t_1 >, ... < e_n, t_n >$, $n \geq 1$, with $e_n =< send, t_s >$. t_s is defined by $t_s = t_r + t_{sw}$ where t_{sw} is the switching time[2]. And (ii) that one or several remote messages $< recv, t_{rr} >$, $t_{rr} = t_s + t_{link}$, are sent to the receiving LP.

The switch model we consider is non-blocking and has output buffers with the following switching strategy: in one time slot each cell that arrives on an input port is switched to the corresponding output queue. The switch logic determines the choice of the output link with the informations stored in a connection table during the connection phase. Only one cell can arrive per time slot on an input but severals cells can be switched to the same output in the same time. In this case they are buffered in the output queue. On the other hand, for all outputs the head cell in the queue is transmitted on the link in each time slot (FCFS server). Upon reception of a $< recv, t_r >$ message, the switch schedules for itself a $< send, t_d >$ event where t_d is the time of departure of the cell at the output queue.

[1] *send* events are generally internal events scheduled upon reception of a *recv* event.

[2] The switching time may be decomposed into several components such as the commutation time, the waiting time and the transmission time.

3.2 The network topology

We propose to study the topology illustrated by Fig. 1 and previously described in the ATM Forum specification of the PNNI routing algorithm [1]. This topology is general with no particular design issues for distributed simulation. We added 7 sources and sinks to model the traffic of cells at the network interfaces. Cells are generated by traffic sources and sent to the corresponding sinks by intermediate ATM switches. A source must send a `setup` message indicating the address of the destination before sending any data cell[3].

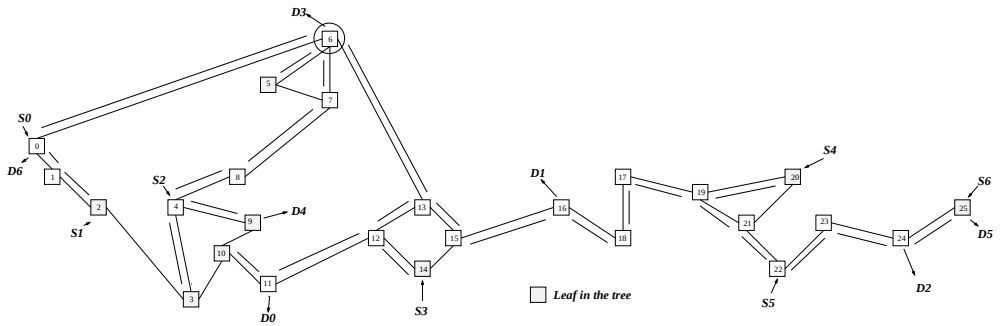

Figure 1: Network model and shortest path to reach D3.

All along the path, a new entry is added in the connection table of each intermediate ATM switch for this new connection. In particular, this table stores the VCI/VPI translation for the connection. Upon reception of the `setup` message, the destination can either send back a `connect` message or a `reject` message. For sake of simplicity, we will assume that the destination never refuse a call demand and that the source can always send cells to its associated sink.

3.3 Routing algorithms

Choosing the best path for the packets/cells from a source to a destination is call the routing function. This is done by considering either implicit criterion such as the number of hops or explicit one given by the user (Quality Of Service). The routing function consists mainly in two phases: (i) constructs new routing tables (done at initialization) and (ii) updates these tables to reflect the network changes (link failures or dynamic routing) [10, 13].

A shortest-path routing protocol can be classified as a *vector-distance* or a *link-state* protocol depending on the nature of the information exchanged between the switches. While routing in packet networks has now a long history, this is not the case for ATM. Since no real large scale ATM network exists, routing in these networks is not well defined yet. Recently the ATM Forum has proposed the PNNI link-state based protocol (Private Network to Network Interface) that should offer scalability, hierarchy, QoS, several routing criterion, dynamic routing and anycast connection [1]. In this paper, we are interested in the routing algorithm proposed by Jaffe and Moss in [8]. This algorithm belongs to the *vector-distance* category and is said by the authors to have a quick recovery from failure property. In Fig. 1, the additional bold links represent the tree that is constructed for a given destination D_3 (the tree is noted $G(D_3)$). Each switch runs a distributed routing algorithm and exchanges cost information to construct a local routing table. At the end of the first phase, each switch is supposed to have a complete table that indicates for each possible destination the best output link. Therefore, for each possible destination D_i a different tree $G(D_i)$ rooted by D_i can be constructed with the branches representing the best link that an intermediate node should take to reach D_i. When the first phase is completed, the switches can begin to send cells from sources to sinks according to their routing table. The second phase consists in experimenting link failures to study the recovery behavior of the routing algorithm. Figure 2 shows the changes in $G(D_3)$ that take into account the two link failures.

[3]We assume only one connection per traffic source.

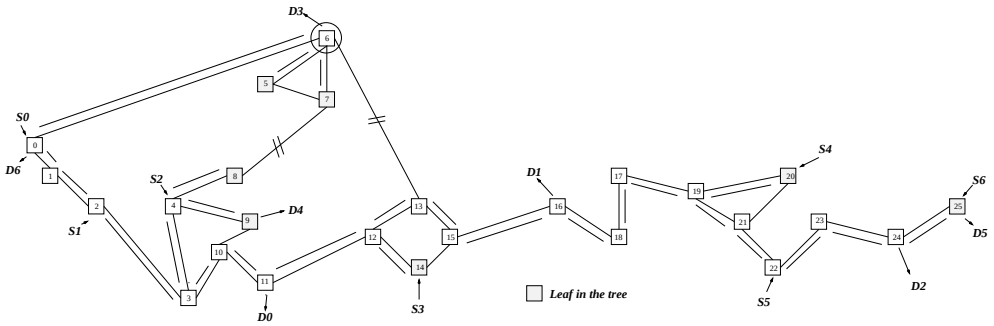

Figure 2: Reconfiguration of the shortest path to reach D3, after failures.

4 Experimentations

Our conservative simulator is written in C++ using an object oriented design. The PVM package (*Parallel Virtual Machine*) [12] provides the communication facilities between processors. The simulation kernel we developed is a unified framework for both sequential and parallel conservative simulation. Therefore, the same source code can either be run sequentially or in a distributed manner. This method has the advantage to allow fair speedup measures, easy debugging and automatic modification of the two versions. A mandatory input file must be provided to the simulator that describes the topology of the network to be simulated. Each LP reads this input file to build the simulated objects. It is possible to specify an optional mapping file that describes the mapping of the network components (switches) into LPs. The default mapping is to assign one LP per network component and to simulate it on one physical processor. With a mapping file, several network components can be grouped in one LP.

The efficiency of our distributed simulator is expressed by comparing the overall execution time of a sequential version running on one node of the parallel machine to the execution time of a parallel version. In a first step, one physical processor simulates one switch of the network model. In a second step, several switches are simulated on the same physical processor with a simple aggregation method that groups several adjacent switches together in the same LP. In this way, we want to reduce the communication cost since switches in the same LP do not need to exchange remote messages nor null-messages. Figure 3 illustrates the 10 processor mapping. In all cases, traffic sources and sinks are never separated from the switch they are connected into.

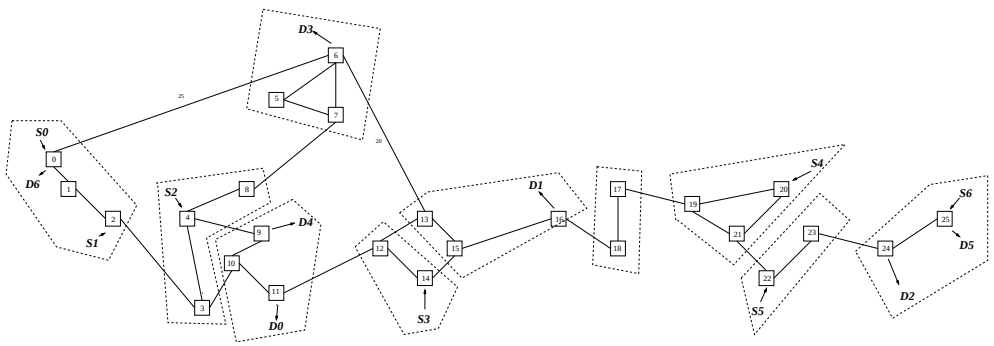

Figure 3: 10 processors mapping.

Figure 4 and 5 show these execution times for the simulation of 100000 time slots on a CM-5 and a Cray T3E respectively. This experiment represents more than 6 millions events simulated (approximately 2 millions cells exchanged between the switches). Table 1 summarizes the speedup measures.

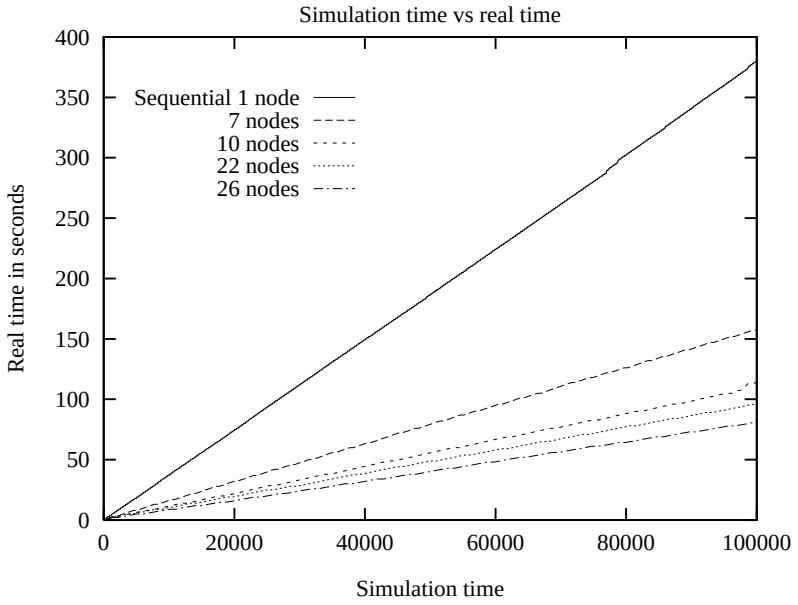

Figure 4: Overall execution time of parallel version on the CM-5.

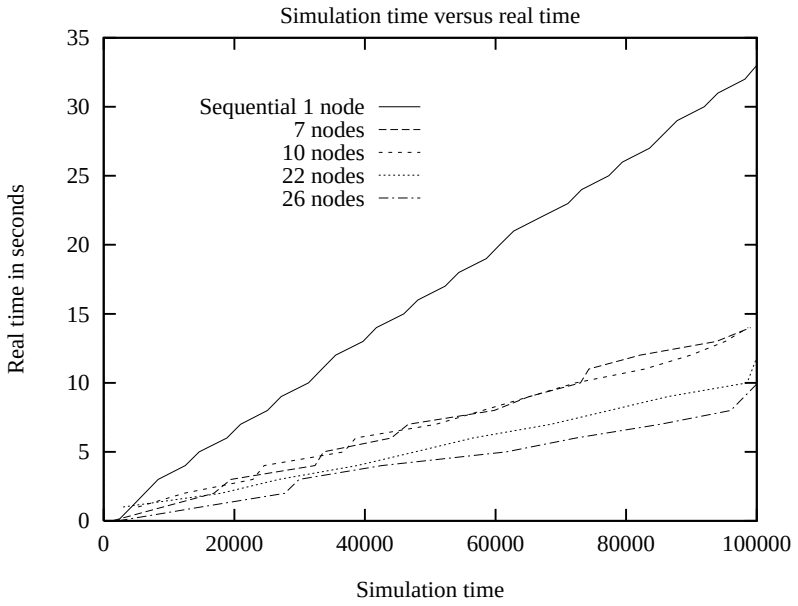

Figure 5: Overall execution time of parallel version on the Cray T3E.

For the one to one mapping, a maximum speedup of 4.52 is achieved on the CM-5. The Cray T3E gives only a speedup of 3.3 because the sequential version runs much better on a more powerful processor. This speedup of 4.52 gives only 0.17 for the efficiency (defined as the ratio between the speedup and the number of processors used). This modest speedup value, when compared to those obtained by previous works, is linked to the complexity of the model. In particular, the routing algorithm, instead of being uniform, creates load imbalances that result in a high null-message ratio between LPs that do not exchange real messages. Also, the general network topology of our tests contributes to lower the degree of parallelism. These figures also show that unfortunately no better speedup can be obtained with less processors. The aggregation method fails to reduce the communication cost between LPs but the reason for this now becomes clear in the light of the results. Let us take the LP that aggregates switches 0, 1 and 2 to illustrate our explanation.

nproc	*real time*		*speedup*		*efficiency*	
	CM-5	T3E	CM-5	T3E	CM-5	T3E
1	380	33	-	-	1.0	1.0
7	158	14	2.41	2.36	0.34	0.33
10	115	14	3.30	2.36	0.33	0.23
22	97	12	3.92	2.75	0.18	0.125
26	84	10	4.52	3.3	0.17	0.126

Table 1: Execution time, speedup and efficiency as *nproc* is varied.

When aggregating adjacent switches on the same physical processor, the number of external links in an LP often remains the same for the case where an LP simulates one switch or for the case where the aggregated LP simulates several switches (2 in the example). This has two consequences. First, the number of null-messages sent by the LP does not change significantly so the overhead for these extra messages is not reduced. Second, the number of external remote messages remains the same since the number of cells sent on a given link is not affected by the logical mapping. The aggregation of switches has the consequence of increasing the load of the processor, because it simulates several switches instead of one, while keeping the same overhead for sending external messages.

Increasing the network size

We now increase the size of the network model in order to look at the scalability of the approach in term of speedup. A network of 78 switches is constructed by taking 3 networks of 26 switches and linking them together. Figure 6 depicts the resulting model. At the moment, it is impossible for us to use more than 32 processors on the Cray T3E so the aggregation method with 32, 16 and 8 processors will be used. Figure 6 also illustrates the 32 processor mapping.

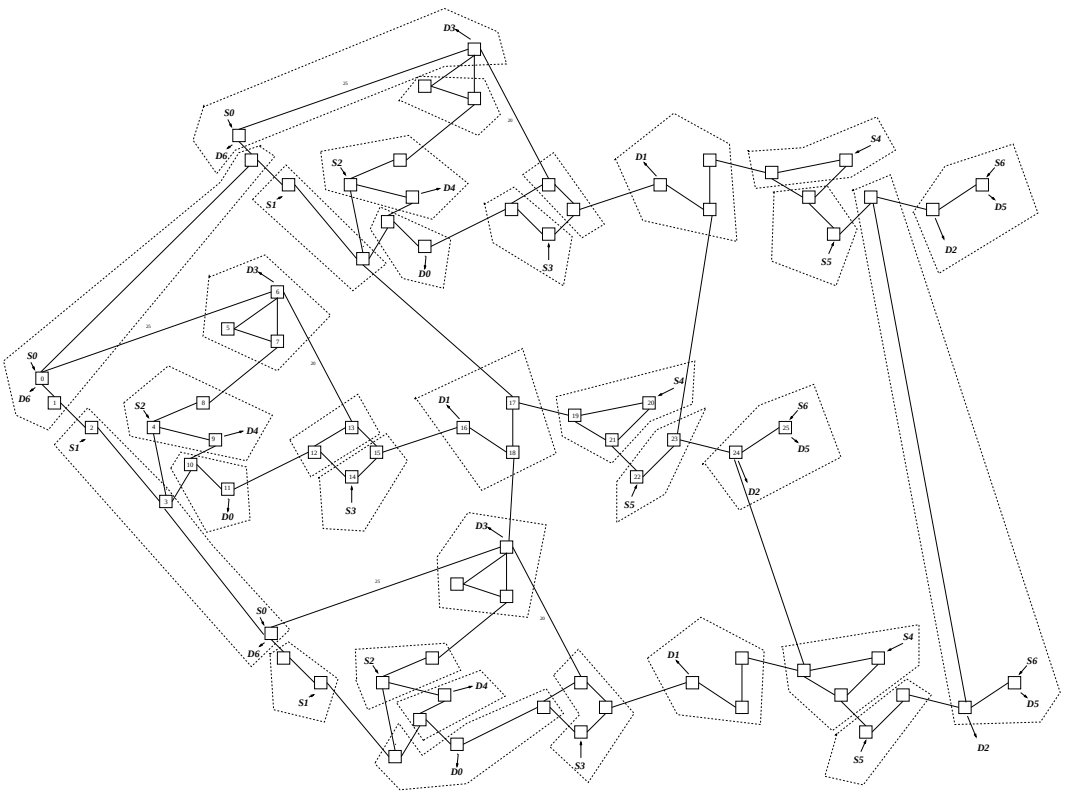

Figure 6: 78 switches network. 32 processor mapping

Figure 7 shows the execution time as the number of processors is varied. Table 2 summarizes the resulting speedup measures. The results show that the maximum speedup is now 7.68 with

32 processors. This is much more than the 3.3 obtained with the previous model. The distributed simulator appears to be less scale-sensitive than the sequential one. In addition, we can observe that the difference between the speedup obtained with 32 processors and the one obtained with only 16 is very small. This phenomenon shows that the aggregation method now allows a better utilization of the processors since the overall speedup is not significantly affected. However, once again, no better speedup can be obtained with less processors.

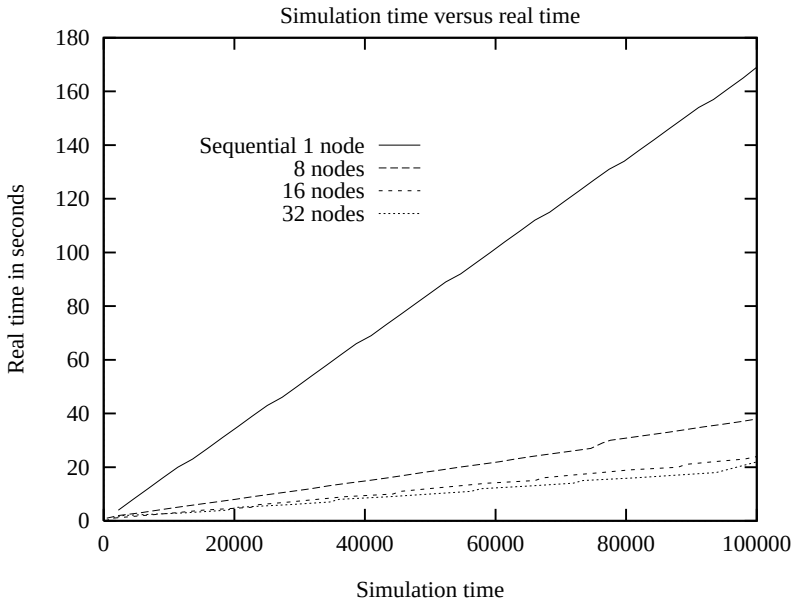

Figure 7: Overall execution time of parallel version when *nproc* is varied.

These results are very encouraging since they clearly show that distributed simulation can handle large configurations and provides significant speedups even on a small number of processors. However, they also point out the difficult problem of optimal partitioning and load balancing in distributed simulation of communication networks. In this particular application where the ratio between real computation and communication overhead (packing/unpacking messages and sending them) is very small, putting several switches that are not adjacent is useless because the level of external synchronization will remain the same. Little gain may only be achieved for small cycles such as the one introduced by switches 5, 6 and 7 (Fig. 3) where we can see that the number of external links is reduced from 4 (case when switch 6 is simulated alone) to 3 (case when switches 5, 6 and 7 are simulated together).

nproc	*real time*	*speedup*	*efficiency*
1	169	-	1.0
8	38	4.44	0.55
16	24	7.04	0.44
32	22	7.68	0.24

Table 2: Execution time, speedup and efficiency as *nproc* is varied.

In most cases, aggregation can improve the efficiency but not the overall speedup. Clearly, a difficult trade-off should be done when aggregating switches between the communication cost reduction and the load increase of the processor. If the computational power of a single processor increases, it should be possible to find a partitioning that maximizes the efficiency while keeping the speedup very close to the one to one mapping speedup.

5 Conclusion and perspectives

In this paper we presented a distributed simulation of a large ATM network with a real routing algorithm. Given a general network topology that consists of 78 switches, a speedup of 7.68 can easily be achieved. The work shows that the use of a parallel computer can help to push the design of large ATM networks because interesting problems can be simulated in a tolerable amount of time. The implementation of additional models would provide a complete framework for performance evaluation of ATM networks with a distributed tool.

Partitioning and load balancing have appear to be essential in distributed simulation. The question is not only *"can we obtain a better speedup with a mapping using less processors than the simple one to one mapping?"* but also *"given a number of processors, can we find the best mapping?"*. Future works would address these problems with more complex partitioning schemes. In particular, it would be interesting to pre-process the network topology in order to extract connection paths to optimize the aggregation.

Acknowledgment

Part of this work is supported by the French defense research institute Direction de la Recherche Et de la Technologie (DRET).

References

[1] ATM Forum, PNNI v1.0 Specification

[2] Chandy, K. M., Misra, J. : Distribution Simulation: A Case Study in Design and Verification of Distributed Programs. Trans. on Soft. Eng., **5**(5) (May 1979) 440–452.

[3] DePrycker, M. : "Asynchronous Transfer Mode : Solution for Broadband ISDN", *Second edition, Ellis Horwood, 1993.*

[4] Earnshaw, R. W., Hind, A. : A Parallel Simulator for Performance Modeling of Broadband Telecommunication Networks. Proc. of the WCS'92 1992, 1365–1373.

[5] Ferscha, A., Tripathi, S. K. : Parallel and Distributed Simulation of Discrete Event Systems. CR-TR-3366, Dept. of Comp. Science, Univ. of Maryland.

[6] Fujimoto, R. M. : Parallel Discrete Event Simulation. Comm. of the ACM, **33**(10) (October 1990) 31–53.

[7] Händel, R., Nuber, M. N., Schröder, S. : ATM Networks: Concepts, Protocols, Applications. Addison-Wesley (1994).

[8] Jaffe, J. M., Moss, F. H. : A Responsive Distributed Routing Algorithm for Computer Networks. IEEE Trans. on Comm., **30**(7) (July 1982) 1758–1762.

[9] Jefferson, D. R. : Virtual Time. ACM Trans. on Prog. Lang. and Sys., **7**(3) (July 1985) 405–425.

[10] McQuillan, J., Richer, I., Rosen, E. C. : The new routing algorithm for the ARPANET. IEEE Trans. on Comm., **28**(5) (May 1980) 711–719.

[11] Mouftah, T., Sturgeon, R. T. : Distributed Discrete Event Simulation for Communications Networks. IEEE JSAC, **8**(9) (December 1990) 1723–1734.

[12] Geist, A. and al. : PVM 3 User's Guide and Reference Manual. (May 1993).

[13] Schwartz, M., Stern, T. : Routing Techniques used in computer communication networks. IEEE Trans. on Comm., **28**(4) (April 1980) 539–552.