Simulation of a Routing Algorithm using Distributed Simulation Techniques

C.D. Pham J. Essmeyer* S. Fdida

Laboratoire LIP6
Université Pierre et Marie Curie
4 place Jussieu 75252 Paris Cedex 05
e-mail : {pham,essmeyer,fdida}@masi.ibp.fr

Abstract. This paper describes our experiment in simulation of a routing algorithm in ATM networks using distributed simulation techniques. These techniques are a promising tool for performance evaluation of large and complex systems that can not be handled sequentially. The simulations are performed on a CM-5 and the results show that interesting speedups can be achieved when compared to a sequential execution. However, they also raise the problem of optimal partitioning and load balancing in communication network models where communication costs represent the main overhead.

1 Introduction

Discrete event simulation often represents the only way to study the behavior of a system. This is because analytical methods may be unsuitable when the size and the complexity of the system 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. However, the flexibility of simulation is paid by a large amount of computation time and a model can turn out very quickly to be very complicated as soon as it is a little realistic. Therefore, there has been a growing interest over the last decade in bringing the available power of parallel machines to the simulation task.

In this work, we report the speedup obtained by distributing the simulation of a complex ATM network model on a Connection Machine (CM-5) of the *Centre National de Calcul Parallèle en Science de la Terre* (CNCPST). Performance evaluation of large communication networks is a challenge because such evaluation requires the execution of millions of events. Distributed simulation of these models is also difficult because the overhead per event must be kept small enough in order to obtain significant acceleration over a sequential simulation. Previous works by Earnshaw and Hind [2], Mouftah and Sturgeon [8] have also investigated the communication network simulation problem but they are limited to the simulation of simple transfer of packet/cell between nodes with uniform

* This work was conducted while the author visited the LIP6 (previously MASI) laboratory for a 6 months period as part of his Master degree.

routing. The speedup reported in [2] 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 [6] and connection management. The simulation of a routing algorithm implies to simulate (*i*) the mechanism that consists in constructing and updating (on cost change such as a link failure) the routing table and (*ii*) the flow of cells transported by the network.

This paper is organized as follows: the next section presents some backgrounds in distributed simulation. Section 3 presents the ATM network model and the routing algorithm. Section 4 details our conservative simulation kernel. The preliminary speedup measures are presented in Sect. 5. Conclusion and future works are then given in Sect. 6.

2 Distributed simulation background

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 (see [3, 4]) but their applications are limited.

The distribution of virtual clocks and event lists requires special care to ensure that the distributed simulation is *correct* and produce the same output when compared to a sequential one. In fact, correctness is achieved if *causality constraints* are maintained. Now, since the different LPs may advance at different rates synchronization problems are likely to occur. For instance, we are confronted to a *causality error* when a message arrives at a receiving LP and is outdated, or old, according to the LVT. These problems arise the need of an explicit synchronization mechanism that are traditionally classified in two categories: *conservative* and *optimistic*.

The conservative approach [1] only processes events that can definitely be considered safe, i.e. that processing an 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* relying on *lookahead* ability in the purpose of artificially propagating and advancing the simulation time. An alternative that consists of detecting the deadlock and breaking it also exists. On the other side, optimistic approaches [7] 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. Instead, a lot of variations on the original scheme are defined to take into account specific problems because the performances of a distributed simulator are tightly linked to the application. In some cases, a distributed simulation may perform worse than a sequential one. 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 The ATM network model

We propose to study the topology illustrated by Fig. 1. Cells are generated by 7 traffic sources and are sent to the corresponding sinks by intermediate ATM switches. Connection mode is also modeled so traffic sources send a `setup` message to their connected switch indicating the address of the destination before sending any data cell.

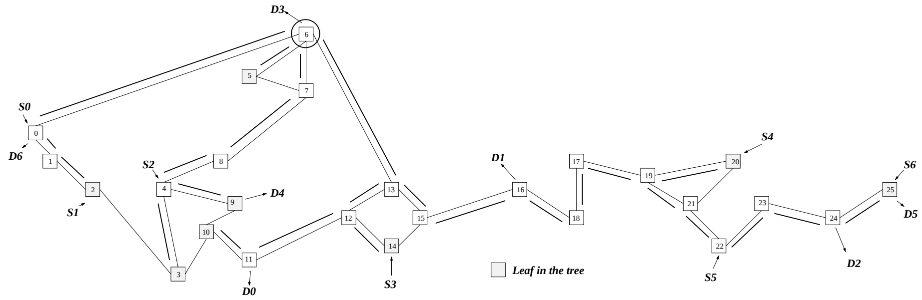

Fig. 1. Routing network topology.

All along the path, a new entry is added in the connection table of each intermediate ATM switches for this new connection. In particular, this table stores the VCI/VPI (Virtual Channel Identifer/Virtual Path Identifer) 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 refuses a call demand and that the source can always send cells to its associated sink.

The routing function consists in calculating the best path for the packets/cells from a source to a sink [10]. This calculation 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 works 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).

In this paper, we are interested in the routing algorithm proposed in [6]. 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 constructed for a given destination $D3$ (the tree is noted $G(D_3)$). The distributed routing algorithm works by exchanging cost information between switches to construct local routing tables. At the end of the first phase, each switch is supposed to have a complete routing table that indicates for each possible destination the best output link. Finally, for each possible destination D_i, a different tree $G(D_i)$ rooted by D_i can be constructed with the edges representing the best link 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.

4 The conservative simulator

The conservative simulator is written in C++ using an object oriented design. The PVM (*Parallel Virtual Machine*) package [9], and especially the CM-5 implementation that uses the underlying low-level CMAML message passing library instead of the high-level CMMD [5], provides the communication facilities between processors. All simulations are performed on a 32 nodes partition.

4.1 The kernel structure

The simulation kernel we developed is a unified object-oriented 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. An abstract base class `CSimulatedObject` is defined for the purpose of deriving child classes for each simulated object such as switch, source and link. In particular, the routing functionality is implemented in the `CRoutingSwitch` class derived from the `CSwitchATM` class that only provides basic switching functionalities. The main part of the simulator is defined in the `CSimulation` class that contains all the simulated objects, the Future Event List (FEL) and the Local Virtual Time (LVT) for an LP. At this point, there is no difference yet between the sequential and the distributed simulation. For conservative distributed simulation, Input Channels (IC) are introduced in order to handle messages from the outside. For this purpose the `CConservativeLP` class is derived from `CSimulation`, and therefore inherits the simulated objects, the FEL and the LVT, and implements the management of the ICs. This means that a sequential simulation only needs the `CSimulation` class while the distributed one uses `CConservativeLP` instead. In both cases, the simulated objects remain the same and they do not know whether they are working sequentially or in a distributed environment.

The management of the ICs and the FEL are the core of the simulator. Since messages arrive on several ICs, one has to scan all the ICs to determine the messages that are safe for processing. Instead of scanning on a per event basis, all messages that are considered safe are transfered to the FEL and transformed into events in one step. Each IC i handles a virtual clock h_i that ticks according to the timestamp of the last received message. In this way, it is possible to determine the minimum of the h_i, called the Sure Future Timestamp (SFT), in order to decide whether a message is safe, i.e. its timestamp is less than the SFT, or not safe, i.e. its timestamp is greater than the SFT. Events produced by an LP are stored inside the FEL that determines whether the destination resides in a different LP on a different processor or in the same LP. This last case can happen when the mapping is not one switch to one processor as explained in the next section.

Null-messages for deadlock avoidance are sent when all safe events have been processed, i.e. the simulation virtual time has advanced to the value of SFT. One null-message is sent on each output link relying on the lookahead provided by the link propagation delay. This delay represents the minimum timestamp increment between 2 successive messages.

4.2 Initialization of the simulation

The programming model for the CM-5 is the master-slave model. At initialization, a master program spawns (with the `pvm_spawn` function) all the LPs on the nodes of the partition and checks for the end of the simulation. The call to `pvm_spawn` must be unique so that all the slaves reside in one CM-5 process in order to achieve fast communication.

A mandatory input file must be provided to the simulator which describe the topology of the network to be simulated. Each slave 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 LP. The default mapping is to assign one LP per switch that is simulated on one physical processor. With a mapping file, several network components can be grouped in one LP. In this case, all objects in the LP are simulated sequentially and conservative synchronization is only needed for external messages.

5 Speedup measures

To measure the efficiency of our distributed simulator we compare 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 attempt, we assign one physical processor to 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. Finally a sequential-like simulation is realized for all the switches in the same physical processor whereas the conservative synchronization mechanism is used between switches on different processor. In this way, we want to reduce the communication cost between adjacent LPs since switches in the same LP do not need to exchange remote message nor null-messages. Figure 2 illustrates the 10 processor mapping. In all cases, traffic sources and sinks are encapsulated in the same LP of the switch they are connected into.

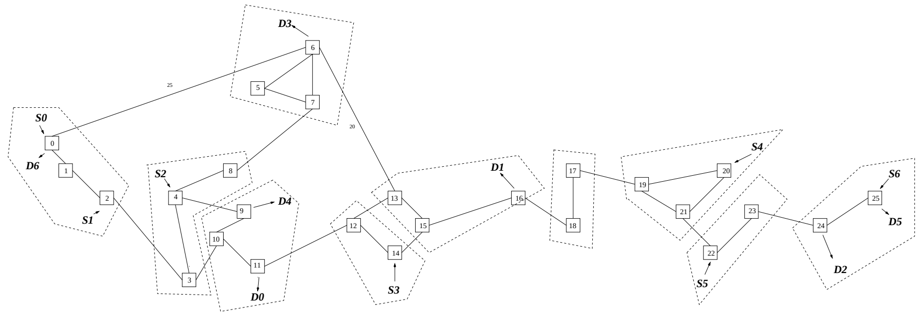

Fig. 2. 10 processor mapping.

Figure 3 shows these execution times for the simulation of 100000 time slots. This represents more than 6 millions events simulated (approximately 2 millions cells exchanged between the switches). For the one to one mapping, the resulting speedup is 4.52. This speedup results in only 0.17 for the efficiency (defined as the ratio between the speedup and the number of processors used). This modest speedup, 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. Figure 3 also shows that unfortunately no better speedup can be obtained with less processors. Therefore it appears that the aggregation method fails to reduce the communication cost between LPs but the reason for these poor performances 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.

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 in-

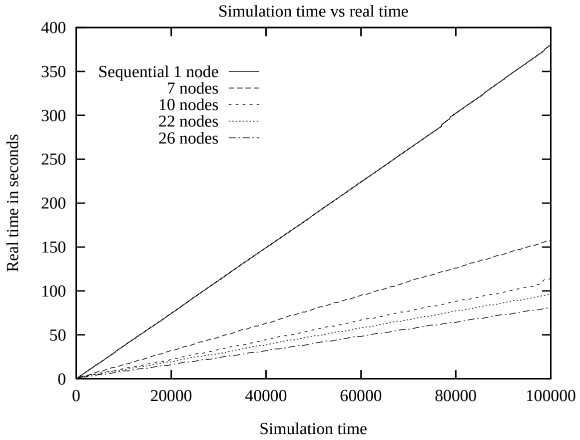

Fig. 3. Overall execution time of parallel version when *nproc* is varied.

stead of one, while keeping the same overhead for sending external messages. These results point out the difficult problem of optimal partitioning and load

nproc	*real time*	*speedup*	*efficiency*
1	380	1.0	1.0
7	158	2.41	0.34
10	115	3.30	0.33
22	97	3.92	0.18
26	84	4.52	0.17

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

balancing in distributed simulation of communication networks. In this particular application where the ratio between real computation and communication overhead (packing and unpacking messages) is very low, putting several switches that are not adjacent is useless because the level of external synchronization will remain the same. Little gain can only be achieved for small cycles such as the one introduced by switches 5, 6 and 7 in Fig. 2 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). 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. On the CM-5, this phenomenon can not be observed because the processing power of a single processor is quite low (SPARC at 32Mhz).

6 Conclusion and perspectives

In this paper we presented a distributed simulation of a real routing algorithm. Given a realistic network topology that consists of 26 switches, a speedup of 4.52 can be achieved with a simple one switch to one processor mapping. Using less processors provides no better speedup since the communication cost remains roughly the same. A speedup of 4.52 is a lot and ... not enough! It is very good because simulations that should take 4 days can be realized in less than a day. Now it is not enough because using 26 processors for a speedup of 4.52 is only 0.17 for the efficiency. However this speedup can be improved: we believe that better speedup can be achieved for larger networks since the parallel version is more scalable than the sequential one. We are currently porting the kernel on a CRAY T3E in order to increase the system size. Nevertheless this work shows that the use of a parallel computer can help to push the design of communication networks because interesting problems can be simulated in a tolerable amount of time.

Acknowledgment

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

References

1. 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.
2. Earnshaw, R. W., Hind, A. : A Parallel Simulator for Performance Modeling of Broadband Telecommunication Networks. Proc. of the WCS'92 1992, 1365–1373.
3. Ferscha, A., Tripathi, S. K. : Parallel and Distributed Simulation of Discrete Event Systems. CR-TR-3366, Dept. of Comp. Science, Univ. of Maryland.
4. Fujimoto, R. M. : Parallel Discrete Event Simulation. Comm. of the ACM, **33**(10) (October 1990) 31–53.
5. Hoppe, H. C., Ossadnik, P., Stüttgen, W. : CM-PVM: An Efficient Implementation of PVM 3.3 for the CM-5. Proc. of The EuroPVM'95, 52–58.
6. Jaffe, J. M., Moss, F. H. : A Responsive Distributed Routing Algorithm for Computer Networks. IEEE Trans. on Comm., **30**(7) (July 1982) 1758–1762.
7. Jefferson, D. R. : Virtual Time. ACM Trans. on Prog. Lang. and Sys., **7**(3) (July 1985) 405–425.
8. Mouftah, T., Sturgeon, R. T. : Distributed Discrete Event Simulation for Communications Networks. IEEE JSAC, **8**(9) (December 1990) 1723–1734.
9. Geist, A. and al. : PVM 3 User's Guide and Reference Manual. (May 1993).
10. Schwartz, M., Stern, T. : Routing Techniques used in computer communication networks. IEEE Trans. on Comm., **28**(4) (April 1980) 539–552.