Optimization of Input Channels: Towards Easy Integration of Conservative Parallelism in Sequential Simulators

C.D. Pham

Laboratoire LIP6
Université Pierre et Marie Curie
4 place Jussieu 75252 Paris Cedex 05
e-mail : Cong-Duc.Pham@lip6.fr
phone: (33-1) 44-27-75-12 fax: (33-1) 44-27-62-86

Abstract

In conservative approaches, the management of the input channels (ICs) can become the bottleneck if the number of ICs dramatically increases. This paper presents a simple method that optimizes the management of ICs (scan and null-messages operations) and reduces the difference between a sequential simulator and a distributed one. Virtual input channels (VICs) are introduced in the conservative approach to achieve this goal. Experimentation that consists in integrating VICs in an existing sequential simulator shows that a distributed version can be obtained with small effort by keeping most parts of the sequential kernel. Also performance results show that the parallel simulation overhead can significantly be reduced with our optimizations.

Keywords: Parallel and distributed simulation, Conservative methods

1 Introduction

Distributed simulation protocols are often classified as either conservative or optimistic depending on the underlying synchronization mechanism. In optimistic approaches [4], a rollback mechanism that uses periodic state saving and anti-messages allows for recovering from time errors. On the other hand conservative protocols [2] introduce input channels (ICs) to store the external events in order to determine the safe events by a scan and wait operation. This method avoids such time errors but null-messages are needed to artificially propagate the simulation time to solve the deadlock problem caused by the waiting on empty ICs scheme. In this case the management of the ICs can become the bottleneck if the number of ICs dramatically increases.

This paper has two goals. The first is to optimize the management of ICs in order to deal with scalability and null-messages in the most transparent manner. The second is to allow for an easy integration of parallelism into existing sequential simulators. This is motivated by the following observation: although the parallel and distributed event simulation (PDES) field is almost twenty years-old now it still remains an academic tool for researchers. As reported in [8], the 1994 PADS panel discussion raised this issue and one reason why PDES has not touch the large simulation community yet was the lack of flexible tools when compared to those used by the majority of users. Even if some parallel simulation tools were specifically developed during the last decade to spread the parallel and distributed words, the gap between these tools, proposed by university people, and those proposed by commercial companies is still so big that every one can understand the preference of users for the latter.

It is not our goal to refuse any specific development because we think that the PDES field has some original characteristics that need to be taken into account in order to achieve full and significant gain. However the advantages of some commercial products should also be reused in a first step. In this case we must try to integrate parallelism with the lowest level of modifications. In that sense, the work presented here goes in the same direction that the U.P.S library described in [8, 9]. The goal of U.P.S. is to offer a user library in order to integrate conservative parallelism to an existing simulation package CSIM [11]. Our work still uses the conservative approach to synchronize distributed simulations but focuses on unifying the queue management in distributed and sequential simulators. A traditional sequential simulator deals with one future event list (FEL) to keep all the messages generated by simulated objects. On the other hand, the conservative method introduces additional queues, called input channels, to keep messages sent by remote processes.

In order to easily integrate conservative parallelism in an existing sequential tool, one should minimize the difference between a conservative kernel and a sequential one. This paper presents a simple technique which reduces the work done by ICs to a minimum in conservative methods. First, some optimizations are done to reduce the scan and the management of null-messages overhead. Finally, we introduce virtual input channels (VICs) instead of ICs. Such VICs can be implemented using only one floating-point variable. The paper is organized as follows: Section 2 reviews the original input channel management, Section 3 presents an optimized approach to manage ICs that avoids the scanning overhead and the storing of null-messages. Section 4 describes the last step that consists of suppressing ICs in order to minimize the difference with a sequential kernel. Integration of VICs in a sequential simulator is then described in Section 5. Section 6 presents the performances results of the proposed approach compared to the traditional IC management. Finally, related works to our approach are presented in Section 7. Section 8 concludes.

2 The original IC management

A simulation can be formalized as a collection of objects that produce and execute events. In a sequential simulation, these objects insert and get events in and from an unique FEL. In a conservative distributed simulation, simulated objects are distributed among a set of Logical Processes (LPs). In this case, events that are scheduled for a distant object have to be sent to the corresponding LP first. Therefore, the communication between LPs requires that static ICs are defined between LPs and that the events on an IC are received in non-decreasing timestamp order. Every IC has an associated clock top_i that keeps the timestamp of the topmost message or, if the IC is empty, the timestamp of the last received message.

Normally, the LP repeatedly searches for the IC i with the smallest associated top_i. If this IC contains an event, the LP can safely process it, otherwise it must block. If there is a cycle of empty queues and they all have the smallest associated top_i of their LP, this synchronization scheme induces a deadlock because all processes in the cycle are blocked. Therefore null-messages are sent by LPs in order to artificially advance the simulation time. A null message with a timestamp t_{null} sent from LP_1 to LP_2 is a promise for LP_2 not to receive a message with a timestamp smaller than t_{null} from LP_1 in the future. Generation of null-messages relies on the possibility for an LP to predict what will happen or not happen in the future (e.g. I won't send a message until T). This feature is commonly referred as *lookahead*.

In this traditional scheduling events are dequeued one by one from ICs. That means that after every scanning of the ICs only one event is dequeued. Of course, as much as possible events are dequeued before the beginning a new cycle of event execution. Also null-messages are stored in the ICs until they are processed.

3 Modified ICs and Safe Future Timestamp

The overhead of IC management consists in 2 parts. The first part represents the scanning operation required to determine the event with the smallest timestamp. This operation is performed until the ICs with the smallest associated clock becomes empty. When the number of ICs is large, the process of repeatedly checking all the ICs can lead to an overwhelming scan overhead. The second part of the overhead is introduced by the null-messages. These artificial messages are sent by remote LPs to advance the simulation time when real messages are not available. Even if the processing of a null-message is only to increase the LP's simulation clock, null-messages still need input buffers and `get` operations from the ICs. Without any enhancement, an IC can receive and store many null-messages from the same LP. Therefore some optimizations have been proposed to reduce the number of null-messages. For instance it is not necessary to store a null-message immediately followed by a real message. However these optimizations also introduce complex code into the simulator.

In a first step we propose a slightly modified version of ICs (referred as MICs) to solve these two problems. First, each IC has a clock $last_i$ that ticks according to the timestamp of the last received message instead of the top_i clock. With this modification, null messages are not stored in the ICs but simply change the value of $last_i$ and do not burden the simulation (figure 1).

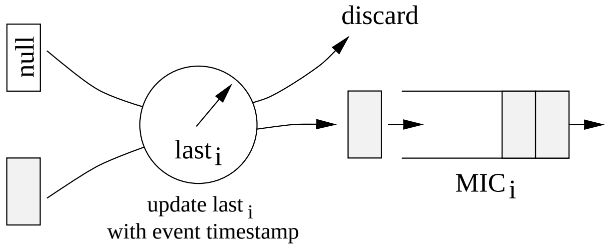

Figure 1: Null message management.

Complex optimizations such as discarding null-messages followed by another messages become obsolete since this is done automatically. Now, if every IC keeps the timestamp of the last received message (real or null-message), it is straightforward to determine the minimum of the $last_i$ in order to define a time window for the LP. We call this minimum Safe Future Timestamp (SFT) and all events smaller or equal to the SFT are safe to process and therefore can be moved to the local FEL to be merged with local events in one step. Figure 2 shows the status of the ICs before and after the scan, where at least one IC should become empty.

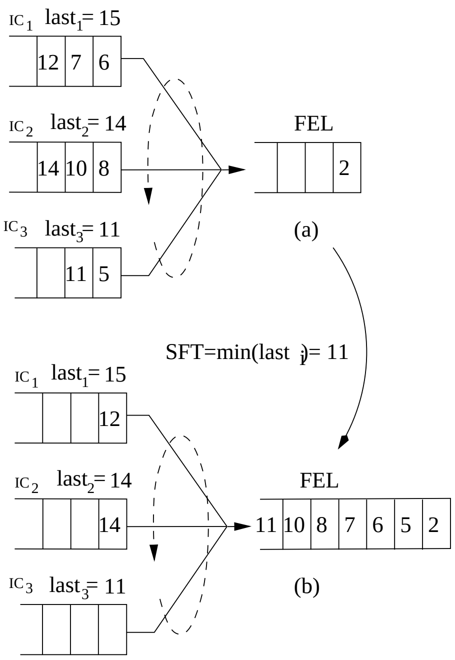

Figure 2: (a) ICs before scanning. (b) ICs after scanning.

4 Virtual Input Channels

A distributed simulation can be regarded as a collection of LPs that interact by means of timestamped messages carrying remote events. The conservative synchronization is only needed at the interface of an LP to determine safe events for sequential processing within the LP. The LP can be decomposed in two parts, (*i*) an FEL that stores all the events for the sequential processing and (*ii*) ICs that temporarily store external events. So the difference between a sequential simulator and a distributed conservative one only resides in the introduction of intermediate buffers to store remote events sent by distant objects and to the generation of null-messages at the end of a processing phase. Local simulated objects to the LP directly interact with the FEL, as in the sequential simulator, and should not even know that they are working with distant objects. Remote messages stay first in ICs waiting for the synchronization checking before being transfered in the FEL like normal events. So we can say that a distributed simulation is a collection of sequential simulators with extra synchronization cost.

The previous modification to ICs offers a better scan operation since all safe events are transfered from ICs to the local FEL in one step. However, this operation still requires the scanning of ICs for events less than the time window. One way to get rid of the scanning operation is to simply suppress ICs. The idea is to use the FELs available in all sequential simulators to store both local events and external events since the SFT guarantees that no messages greater than the SFT are processed. This is possible because no artificial messages are stored in the MIC scheme anymore.

This new step reduces the work done by ICs to a minimum since it appears now that the difference between a sequential and a distributed simulation is only the upper bound of the time window. In a sequential simulation, the SFT can be set to the time of the end of the simulation whereas in a distributed simulation, it progressively advances according to the arrival of external events for synchronization purpose. Therefore, the ICs should only be introduced in order to determine this SFT. In other words, an IC can now be simply a floating-point variable that stores the timestamp of the last received message for this IC. That is why we call our ICs Virtual Input Channels (VICs). After keeping the timestamp of an incoming event in the VIC, this event can directly be stored in the FEL waiting for the time window to grow until its timestamp. This approach has the following advantages to be mentioned:

- VICs are no longer represented by queues. Instead, simple floating-point variables which are far less complex than ICs are used. Reducing complexity of ICs also reduces the amount of modifications to an existing sequential simulator.

- There are no scanning operations anymore since external events are stored directly in the

FEL like local events. The management of ICs is now reduced to a minimum without multiple transfers of events in memory. In particular, this approach should be more scalable than the traditional scanning method.

- The waiting on an empty IC is now a waiting for the SFT to increase. Therefore no tests on empty ICs are needed.
- Once the external events are put in the FEL, nothing can distinguish them from local events. Therefore the integration of parallelism into sequential simulators is easy and the sequential kernel for event processing can be kept unchanged.

5 Integrating VICs in a sequential simulator

We have integrated VICs to a sequential simulator written for simulation of ATM network models. Our simulator has an object-oriented design but the modifications can be easily extended to non object-oriented simulator code. However, the object hierarchy provides an elegant integration of parallelism by inheritance. This should minimize the amount of new code without changing the old one. We can roughly describe our sequential simulator (`CSimulation` class) by three main methods as follows:

```
CSimulation::InsertInFEL(CEvent& ev)
{  m_FEL.insert(ev); }
```

```
CSimulation::Run()
{ ProcessFELUntil(tsEnd); }
```

```
CSimulation::ProcessFELUntil(CTimeStamp SFT)
{  while ( LVT < SFT)
   {  if ( not GetNextEvent(ev)) break;
      LVT = ev.GetTimestamp();
      SimObjects[ ev.GetObjectId() ].ProcessEvent(ev);
   } }
```

The `InsertInFEL` is the standard method that insert an event in the future event list. Events are automatically sorted by their timestamp. The `ProcessFELUntil` sequentially processes the events in the FEL until a given time. Finally the `Run` method starts the simulation. For the integration of parallelism, we introduce the `CConservativeLP` class that inherits from `CSimulation`. It therefore contains everything a sequential simulation contains (i.e. local simulated objects, LVT, FEL, `InsertInFEL`, `Run` and `ProcessFELUntil` methods). Additionally it maintains the VICs. Figure 3 illustrates the global view of the conservative LP.

One main additional method, called `GetMessagesFromBuffer`, is introduced that collects the messages from the environment buffers (PVM or MPI for example) to the LP's memory.

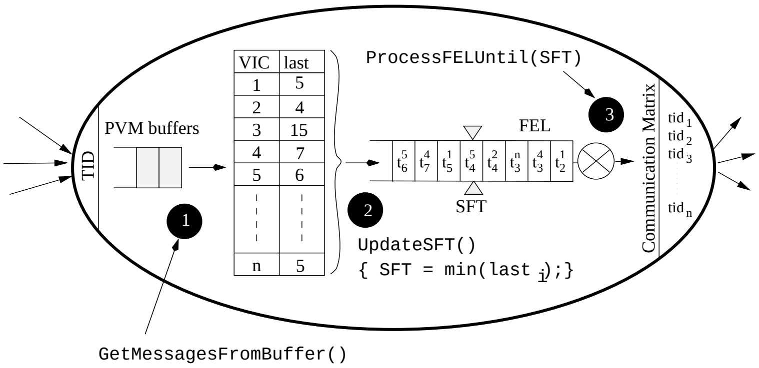

Figure 3: Structure of a conservative LP.

The `Run` method of `CSimulation` is overloaded because the logical process is not allowed to simulate until the simulation end time without synchronization. It has to handle the received messages and the null-message generation. Also the `InsertInFEL` method should now distinguish between local and remote events that should be sent to other processes.

```
CConservativeLP::GetMessagesFromBuffer()
{  while ( SomeMessagesLeft )
   {  nMessage++;
      GetMessage(ev);
      last[ev.GetIC()] = ev.GetTimestamp();
      if ( ev.GetType() != EV_NULL )
        CSimulation::InsertInFEL(ev);
   }
   return ( nMessage ); }
```

```
CConservativeLP::Run()
{  SendNullMessages(0.0);
   while ( not Finished )
   {  while ( GetMessagesFromBuffers() == 0) ;
      if ( UpdateSFT( SFT ) )
      {
        ProcessFELUntil( SFT );
        SendNullMessages( LVT );
      }
      Finished = HandleStoppingProtocol();
   } }
```

```
CConservativeLP::InsertInFEL(CEvent& ev)
{  if ( isLocalEvent(ev.GetObjectId() )
    CSimulation::InsertInFEL(ev);
   else
     Send(ev); }
```

The `GetMessagesFromBuffers` method moves the messages from the reception buffer of the environment directly into the FEL of the sequential simulator, maintains the VICs by updating the $last_i$ variable (step 1 of fig. 3) and discards the null-messages. This is a very simple method. The `UpdateSFT` method determines a new SFT by taking the minimum of the $last_i$ (step 2 of fig. 3). This is a straightforward method that returns `true` if the SFT has increased. `ProcessFELUntil` processes all events from the FEL until SFT. This is the unchanged method of the sequential simulator (step 3 of fig. 3). After the processing of all events until SFT (not only the new received but also the local events which have been in the FEL before) the LP sends one null-message on every output channel. The `HandleStoppingProtocol` takes care for the distributed termination.

A line number with a star indicates the use of the function available in the sequential simulator. For a general commercial sequential simulator, we can reasonably assume that 2 methods equivalent to the `InsertInFEL` and the `ProcessFELUntil` methods are provided. Therefore, the integration of parallelism, under the assumption that communication facilities are provided, can be obtained at a very low cost without changing the structure of the sequential kernel.

6 Performance study

In order to test the performance of our optimizations, we compare the simulation time of a test network under the standard IC management (STD), the modified one (MIC) and the virtual IC concept (VIC). Since we use a conservative simulator of ATM network models developed at the LIP6 Laboratory we will speak in term of switches and cells. The test network consists of a 4-switch horizontal line, switches from number 0 to 3, with 2 h-switch vertical lines as depicted in figure 4. The number of switches depends on the height h of the network as follows: $n_s = 2h + 4$. A source of traffic, connected to switch 0, sends cells to a sink connected to switch 3. The inter-arrival time between 2 cell arrivals is exponentially distributed. The link propagation delay is set to 3 time units (tu) for all the horizontal links and to 5 tu for the others. The underlying routing algorithm takes this value as a static cost for the link. Therefore all the cells from the source are routed along the horizontal line since this path has the lowest cost (9 tu instead of 10 tu).

The simulations are performed on a Cray T3E with 256 processors located at the IDRIS institute (*Institut du Développement et des Ressources en Informatique Scientifique*). Each switch i of the network is simulated by an LP i on a dedicated physical processor. The source and the sink are put in the same LP of their associated switch. At the moment, no more than 32 processors can be used in interactive mode on the Cray. Therefore the maximum height for the network is 14 resulting in $2 * 14 + 4 = 32$ processors. By varying the value of h, it is possible to dramatically increase the number of ICs for the horizontal switches. The impact on the conservative simulator is two-fold: (i) since cells are only routed along the horizontal

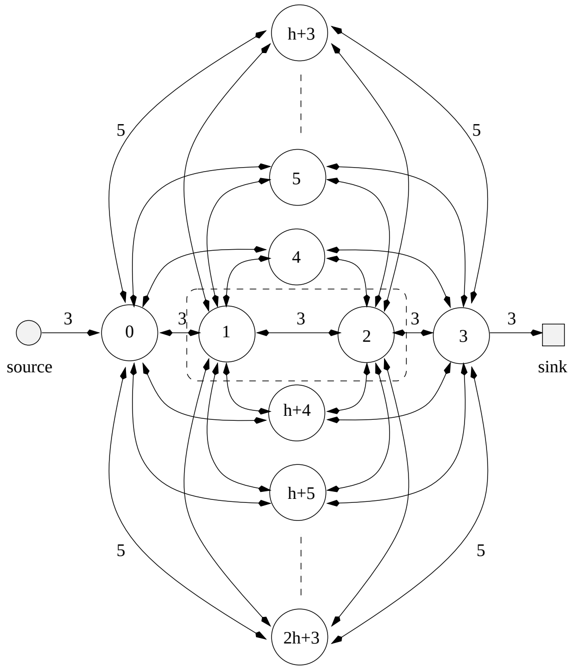

Figure 4: The test network with height h.

links a lot of null-messages are needed for the other links, and (ii) the large number of ICs make the scanning overhead the most important overhead. The link propagation delay is used as the lookahead value by the simulator. There is no attempt to enhance this lookahead value because we want to test for the most general case and to provide the highest transparency.

In figure 5 we plotted the null-message ratio (NMR), defined as the ratio between the number of null-messages received and the total number of messages received, as the number n_{ic}^1 of ICs for LP_1 (that simulate switch 1) is varied. n_{ic}^1 depends on the height h of the network as follows: $n_{ic}^1 = 2h + 2$.

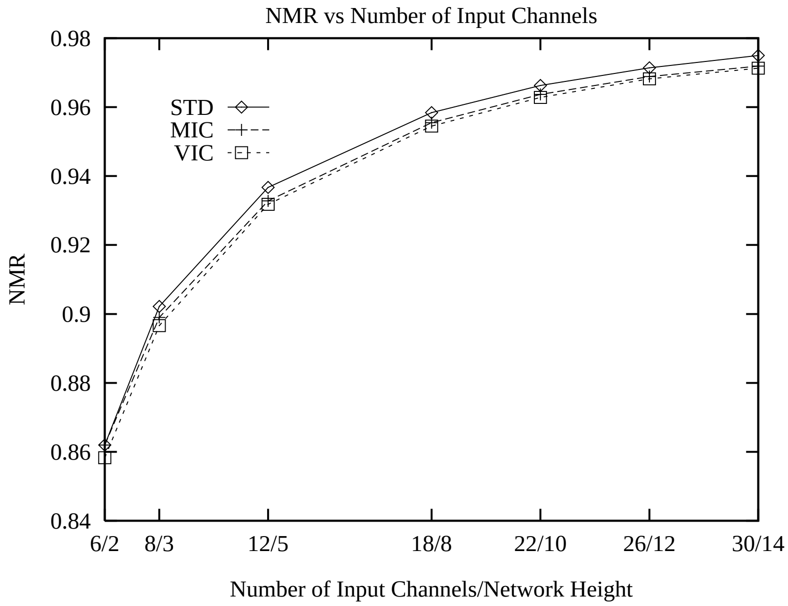

Figure 5: Null Message Ratio (NMR) of STD, MIC and VIC schemes for LP_1.

For instance, a network height of 14 leads to 30 ICs for LP_1. We can see that the NMR is very high: 86% for 6 ICs to 97% for 30 ICs. There are no significant differences between the STD, MIC and VIC management even if the last scheme provides the lowest NMR because of a slightly better improvement of the cycle size. This was foreseeable since the optimizations are not dedicated to reduce the number of null-messages but rather to lower their overhead. However, it was important to show that the NMR will not contribute to the decrease of the simulation time in the next tests. Figure 6 shows the simulation time as a function of the number of ICs. We can see that the VIC scheme can reduce this execution time about 40% . This improvement is mainly due to the improvement of the scanning operation and to the automatic discard of null-messages since the execution times for MIC and VIC scheme are very close. The VIC scheme is a bit better because events are directly inserted in the FEL instead of temporarily staying in the input channels.

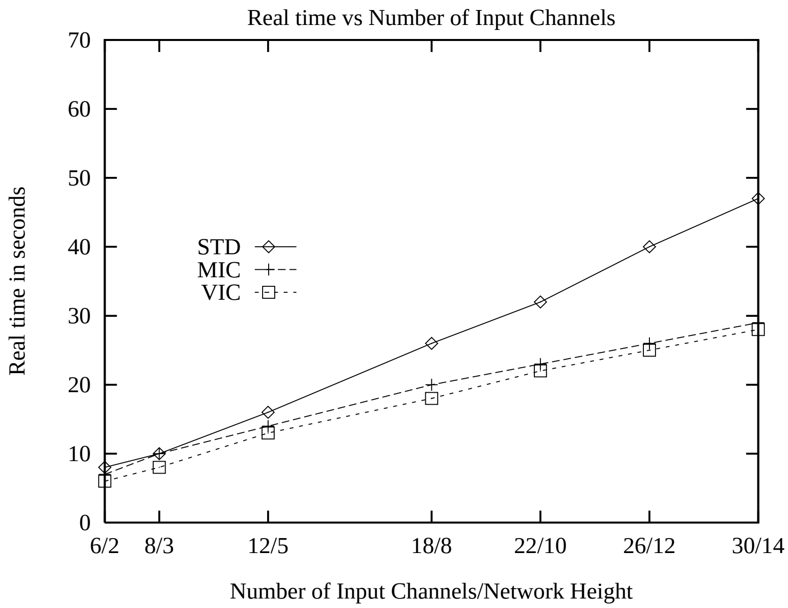

Figure 6: Simulation time of STD, MIC and VIC schemes.

In order to artificially increase the number of ICs, we now aggregate switch 1 and 2 into the same LP (LP_1) as depicted by the dashed square in figure 4. Therefore, the number of ICs for LP_1 is now $n_{ic}^1 = 4h + 2$ instead of $2h + 2$. The NMR is found very close to the previous case. Figure 7 shows the execution time as n_{ic}^1 is varied for the aggregated case. We can see that the decrease of the simulation time between the different ICs management schemes is smaller than the previous case for the same approximative value of n_{ic}^1. At the moment the reason for this is not very clear. Nevertheless, this test shows that the simulation time can be cut in half with the VIC scheme for a very large number of ICs. It does seem not reasonable to increase the number of ICs any further since applications that exhibits so many ICs per LP are hard to find.

All our tests show that both MIC and VIC scheme are much more scalable than the original IC management strategy when the number of ICs becomes large. Even if the network model is not realistic in terms of load, the number of links a switch can have in a backbone ATM network may be close to the one of our model. Another benefit of our optimizations is to increase the distributed simulator overall performance without decreasing the level of transparency for the end user. For instance, the need to have a very application-specific null-message generation strategy, to deal with complex loops and to detect unused links is reduced with the proposed optimizations. This is particularly useful for communication network models. Now, the main contribution of VIC over MIC, if not from the performance side, is to minimize the differences between a sequential simulator and a distributed conservative one.

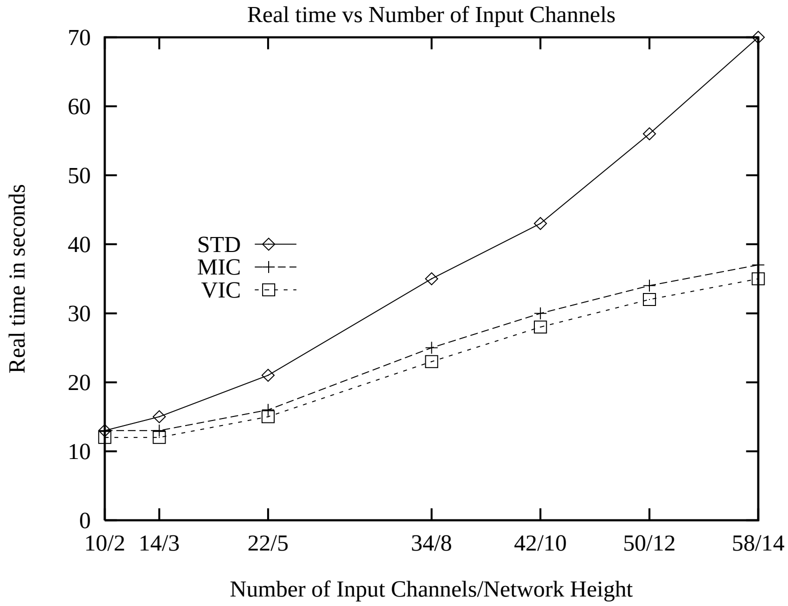

Figure 7: Simulation time of STD, MIC and VIC schemes. Aggregated Switches

7 Related works

The work presented here can be related to the U.P.S. package proposed by Nicol and Heidelberger [8, 9] since the goal is to allow easy integration of parallelism in existing software. However, U.P.S is a library package for easy integration of parallelism into the CSIM language whereas our approach proposes at a lower level a simplified management of ICs for the conservative approach.

Time windows have been used in both optimistic and conservative approaches for different purposes. In conservative approach such as *Bounded Lag* proposed by Lubachevsky [6], the notion of *minimum propagation delay* between 2 LPs is introduced in order to determine a safe time window for event processing without conservative blocking. The algorithm is synchronous and needs a broadcast mechanism to calculate the global time of the simulation. *Conservative*

Time Window [1] is quite similar to *Bounded Lag* but tries to determine a set of time windows W_i such as events in each W_i are independent. Also, cooperation is required between LPs to find the size of the W_i. Time windows in optimistic approaches have the main purpose to limit optimism. *Moving Time Window* proposed in [10] uses the time window to introduce also 4 temporal positions for an event depending on its timestamp. Specialized recovery schemes other than the rollback mechanism can then be used with some type of events. *Adaptive Time Ceiling* [7] uses a dynamic window size that is determined by the time error frequency. Therefore the time window limits the progression of faulty LPs. We use a time window in a very simple manner. Our time window defined by the value of the SFT reflects the local knowledge of safe events relying on the original conservative "blocking on empty queue" paradigm. However, the blocking mechanism is hidden behind the abstract view of waiting for the SFT to increase.

Optimizing ICs scanning is not a new idea. It has already been used in many conservative implementations. Recently, Cleary and Tsai have proposed a variant that consists in maintaining an EmptyList for empty ICs in order to optimize the merge operation [3]. Our solution to directly insert external events without any test in the local FEL provides minimum management of ICs. Jha and Bagrodia have proposed a unified view for conservative and optimistic approaches [5]. In their proposition, for each LP they define an *Earliest Output Time* (EOT) and an *Earliest Input Time* (EIT). The EIT for LP i is defined as the minimum of all the EOT sent by LP j that communicate with LP i. This notion of EIT is similar to the SFT described in this paper. Cleary and Tsai also used a similar concept for their merging algorithm [3]. However, the way we use the SFT with VICs is, at the best our knowledge, new. Also the management of null-messages in the way we described differs from other implementations.

8 Conclusion

This paper has presented a simple method to reduce the work done by input channels in the conservative approach to a minimum. A modified version of ICs uses a $last_i$ clock instead of a top_i clock to automatically discard null-messages. Going one step further leads to the definition of a Virtual Input Channel which is now just a floating-point variable that keeps the timestamp of the last received message (real or null) on the IC. External events and local events are then in the same FEL but external events are discarded for execution by a time window. This approach reduces the scanning overhead and avoids the storing of null-messages in the ICs. Furthermore, it leads to a very simple implementation for ICs. The experimentations show that our optimizations can reduce the overall execution time about 50% when the number of ICs is large.

As said in [8] if the cost of extending parallelism to sequential tools can be reduced to a minimum, it would be attractive for some companies to integrate parallelism to their commercial simulators. The concept of VICs does reduce the differences between a sequential simulator and a distributed one to only an array of floating-point variables and communication facilities between LPs. We hope that the amount of modifications is now less enough to incite commercial companies to offer some parallelism facilities in their simulators.

References

[1] R. Ayani and H. Rajaei, "Parallel Simulation Using Conservative Time Windows", *Proceedings of the 1992 Winter Simulation Conference, pp709-717.*

[2] K. M. Chandy and J. Misra,"Distribution Simulation: A Case Study in Design and Verification of Distributed Programs", *Transactions on Software Engineering, Vol. 5(5), May 1979, pp440-452.*

[3] J. G. Cleary and J. J. Tsai, "Performance of a Conservative Simulator of ATM Networks", *Proceedings of PADS'97, June 1997.*

[4] D. R. Jefferson, "Virtual Time", *ACM Transaction on Programming Languages and Systems, Vol. 7(3), July 1985, pp405-425.*

[5] V. Jha and R. L. Bagrodia, "A Unified Framework for Conservative and Optimistic Distributed Simulation", *Proceedings of 8th Workshop on Parallel and Distributed Simulation (PADS'94), pp12-19.*

[6] B. D. Lubachevsky, "Efficient Distributed Event-Driven Simulations of Multiple-Loop Networks", *Communication of the ACM, Vol. 32(1), January 1989, pp111-123.*

[7] Y. Matsumoto and K. Taki, "Adaptive Time-Ceiling for Efficient Parallel Discrete Event Simulation", *Proceedings of the 1993 Object-Oriented Simulation Conference, pp101-106.*

[8] D. Nicol and P. Heidelberger, "On Extending Parallelism to Serial Simulators", *Proceedings of 9th Workshop on Parallel and Distributed Simulation (PADS'95), pp60-67.*

[9] D. Nicol and P. Heidelberger, "Parallel Execution for Serial Simulators", *ACM Transactions on Modeling and Computer Simulation, Vol. 6(3), July 1996, pp210-242.*

[10] L. M. Sokol, B. K. Stucky and V. S. Hwang, "MTW : A Control Mechanism for Parallel Discrete Simulation", *Proceedings of the International Conference on Parallel Processing, 1989, pp250-254.*

[11] H. Schwetman, "CSIM: A C-based, Process Oriented Simulation Language", *Proceeding of the 1986 Winter Simulation Conference, pp387-396.*