Optimizing Geographically Distributed Timed Cosimulation by Hierarchically Grouped Messages*

Sungjoo Yoo Kiyoung Choi

Design Automation Laboratory
School of Electrical Engineering
Seoul National University
Seoul 151-742, Korea
{ysj,kchoi}@poppy.snu.ac.kr

Abstract

This paper presents a concept called **hierarchically grouped message** *to improve the performance of geographically distributed timed cosimulation. In the proposed method, messages which are transferred between simulators in a short period of simulated time are hierarchically grouped into a physical message to reduce the number of rollbacks in optimistic simulation as well as the communication overhead of message transfer. Experiments show the efficiency of the proposed method in an internationally distributed cosimulation environment.*

1 Introduction

In geographically distributed cosimulation environments, designers can simulate a system which consists of various remotely located intellectual property (IP) blocks without requiring local copies of the IP blocks. IP providers and EDA vendors also have benefits of allowing their IP blocks and proprietary tools, e.g. high performance hardware emulators, to be accessed while protecting their intellectual property rights.

However, high communication overhead in geographically distributed cosimulation environments prevents designers from performing detailed timed cosimulation of communication intensive systems. The problem gets more serious when *interrupt* is used as the communication protocol in the system being designed, since hardware and software simulators should synchronize with each other (via slow communication over Internet) at every system clock tick to detect the occurrence of interrupt [1].

There have been few researches on optimizing geographically distributed timed cosimulation. As an optimization method, [2][3] present a method, called *selective focus*, which dynamically changes the abstraction levels of communication models to allow designers to trade off between performance and accuracy. Contrary to [2][3], we present an optimization method which preserves the accuracy of detailed cosimulation.

In this paper, we focus on geographically distributed timed cosimulation of systems having interrupt as one of communication protocols. By a geographically distributed cosimulation environment, we mean a network of workstations (or PC's) over Internet (or Wide Area Network). Basically, our approach to the reduction of simulator synchronization overhead (including communication overhead) is to apply optimistic simulation concept to geographically distributed timed cosimulation since optimistic simulation is advantageous especially when communication overhead is dominant [1][4][5].

However, since communication overhead is excessive in geographically distributed cosimulation environments, the performance gain obtained by applying conventional optimistic simulation methods can be limited. In applying optimistic simulation to geographically distributed timed cosimulation, the effects of such high communication overhead on the increase of cosimulation run-time are twofold: (1) excessive rollbacks, i.e. rollback overhead caused by the slow transfer of messages compared to the simulation execution as well as (2) the communication overhead itself. According to our experiments, optimistic simulation suffers from excessive rollbacks when intensive synchronization between simulators is performed in a short period of simulated time. It is because while messages are being transferred via slow communication over Internet, the optimistic simulator that is to receive the messages runs further into the future, which causes rollbacks in the receiving simulator. To reduce such excessive rollbacks and high communication overhead, we present a concept called *hierarchically grouped message* (HM) where messages transferred between simulators in a short period of simulated time are hierarchically grouped into a physical message.

This paper is organized as follows. In Section 2, we give a brief description of applying optimistic simulation to timed cosimulation. Section 3 explains our motivation. We present hierarchically grouped message concept in Section 4. We give experimental results in Section 5. Section 6 concludes this paper.

2 Background

In this section, we describe three types of timed cosimulation considered in this paper: uni-processor synchronous cosimulation, hybrid cosimulation, optimistic distributed cosimulation. In the following, we define a *message* to be a timestamped event.

2.1 Uni-processor Synchronous Cosimulation

In Figure 1, we assume that software (SW) and hardware (HW) start to run concurrently at time 0. The exact time when HW sends an interrupt to SW is not known *a priori* but given as a time interval. In Figure 1, blank rectangles and numbers on them represent simulation workloads and the corresponding local times in the simulator they are running on, respectively. Blank arrows represent *null messages* for simulator synchronization only and shaded arrows represent interrupts from HW to SW. Shaded rectangles represent **simulator synchronization overhead** in cosimulation runtime. In *synchronous cosimulation* as shown in Figure 1 (a), SW

*This work was supported in part by ETRI, Korea.

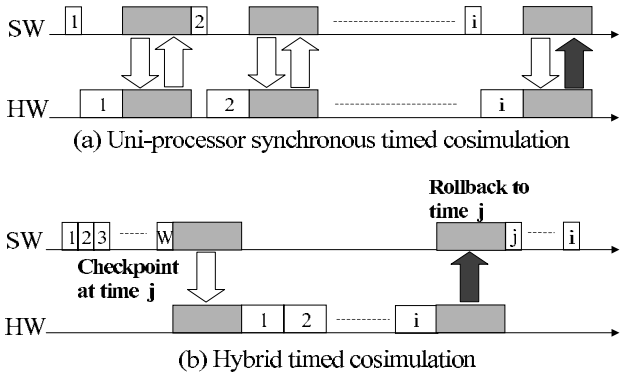

Figure 1: Reduction of synchronization overhead by hybrid cosimulation.

and HW simulators synchronize with each other at every system clock tick to detect the occurrence of interrupt. In synchronous cosimulation, most of simulation run-time can be consumed in simulator synchronization [1].

2.2 Hybrid Cosimulation

Most simulators do not support optimistic simulation features such as checkpoint (or state saving) and rollback. The same is true for emulators which can hardly support a cycle-accurate state saving feature. Therefore we consider a hybrid cosimulation where optimistic simulator(s) and synchronous simulator(s) (including emulators) co-exist. A representative case of the application of hybrid cosimulation is accessing a high performance HW emulator via Internet.

In *hybrid cosimulation* [1] as shown in Figure 1 (b), to reduce simulator synchronization overhead, we first run the optimistic simulator for a time window of predetermined size **W**. In this example, we assume that the SW simulator is an optimistic simulator. The optimistic simulator stops after the time window **W** elapses or at a time point **W'** ($<$ **W**) when a message is sent to the synchronous simulator (in this example, the HW simulator), and waits for messages from the synchronous simulator. During the simulation, states of optimistic simulation are stored at checkpoints in preparation for the rollback. The synchronous simulator starts to run until the time point when the optimistic simulation stops. The synchronous simulation may stop earlier if the synchronous simulator sends a message to the optimistic simulator. In this case, since the timestamp of the message sent to the optimistic simulator is earlier than the time point when the optimistic simulator has stopped, the optimistic simulator rolls back to a checkpoint before (or equal to) the timestamp of the message. If there is no message from the synchronous simulator to the optimistic simulator, then the synchronous simulator stops at the time point **W** (or **W'**). After determining a new **W**, the optimistic simulator starts to run until **W**. Then, the cosimulation continues in this way. Note that in hybrid cosimulation a simulator stops its simulation when it sends a message to another simulator or after the time window **W** or **W'** elapses.

2.3 Optimistic Distributed Cosimulation

In optimistic distributed cosimulation, a set of *logical processes* (physically, optimistic simulators) execute concurrently and communicate by exchanging messages. A logical process (LP), as a unit of parallel simulation, consists of (1) the simulation model of the sub-system being simulated, (2) a state queue to store the states of the simulation model, (3) an input message queue for messages which arrive at the LP, and (4) an output message queue for messages which the LP sends to other LP's. Each LP has its own local time called *local virtual time* (LVT). Each LP works as follows. After advancing LVT, the LP looks up the input message queue to find an input message having a timestamp equal to LVT, processes the message, and advances its LVT. If there is any unprocessed input message which has a timestamp earlier than LVT (we call such a message a *straggler message*), the LP rolls back its LVT according to the timestamp of the straggler message, i.e. the state stored at the time point earlier than or equal to the timestamp of the straggler message is restored.

To support rollback, states are stored at checkpoints. To constrain the memory usage of simulation host for state saving, a *global virtual time* (GVT) is calculated. GVT is the minimum of timestamps of *in-transit messages*[1] and local virtual times of all LP's. States and messages having timestamps earlier than GVT can be removed from the state queue and the input/output message queues.[2] For more details on optimistic distributed cosimulation, refer to [5]. A representative case of applying optimistic distributed cosimulation is accessing the simulation models of IP blocks and performing their simulations via Internet.

3 Motivation

Grouping multiple messages into fewer numbers of physical messages gives faster transmission of messages than transmitting each message separately, since the communication overhead over Internet does not strictly depend on the sizes of messages being transferred, but rather strongly depends on the number of physical messages transferred

Grouping messages also has the advantage of reducing the number of rollbacks. Figure 2 illustrates an example of communication of messages between a SW simulator and a HW simulator in optimistic distributed timed cosimulation. In Figure 2, we assume that SW receives 64 data from HW. In SW processors, such a communication can be performed by executing memory load instructions (e.g. **LDR** or **LDM** instructions in ARM7 processor [6]). To receive each of the data, SW sends the address value to HW (event on the address bus). HW sends the datum corresponding to the received address value to SW (event on the data bus). In memory load (or store) instructions, the time gap between the event on the address bus and the event on the data bus is within a few clock cycles in the simulated time.

However, due to high communication overhead (e.g. at least a few milliseconds per message transfer) in geographically distributed cosimulation environments, when the datum requested by SW arrives at the SW simulator, the SW simulator (one which has millions cycles/sec performance on high performance workstations) may have proceeded further into the future in the simulated time. Such a straggler message causes rollback in the receiving optimistic simulator, in this example, the SW simulator. Figure 2 also illustrates rollbacks (upward arcs) caused by such straggler messages. As shown in Figure 2 (a), optimistic distributed timed cosimulation suffers from excessive rollbacks when intensive synchronization between simulators is performed in a short period of simulated time.

To reduce such excessive rollbacks, we use a *hierarchically grouped message* (HM) concept. Figure 2 (b) shows simulator synchronization using HM's. In this example, we group 64 messages transferring from SW to HW (HW to SW) into a single physical message **HM2hw** (**HM2sw**). In constructing a new physical message, we neither merge original events into a new event nor increase

[1] Messages which are in the communication channels between LP's, or not processed yet in input message queues. In our implementation, the Internet communication channel works as a FIFO queue.

[2] If there is no state stored at GVT, the state having the latest timestamp (but earlier than GVT) is kept in the state queue.

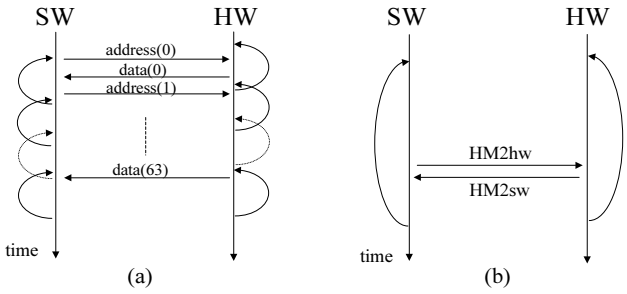

Figure 2: Reduction of rollbacks by hierarchically grouped messages.

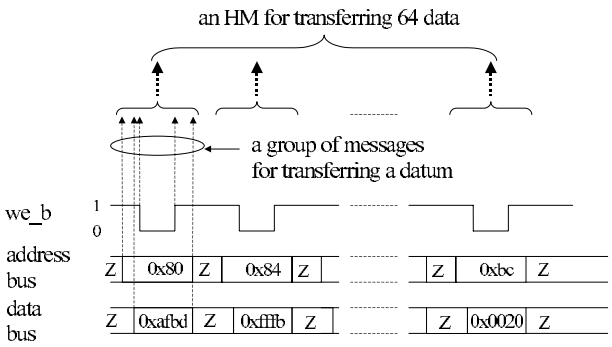

Figure 3: An example of hierarchically grouped message.

the abstraction levels of messages, i.e. original events (i.e. their abstraction levels) are kept unchanged in our method. Since only a single physical message is sent from SW (HW) to HW (SW) in Figure 2 (b), rollback occurs only twice in total.

Such reduction of excessive rollbacks, however, does not come for free. Since the transfer of messages is delayed for the construction of the whole HM, *rollback distance* (the amount of simulated time which is canceled by rollback) on each simulator may increase. In Section 5, however, we show experimentally that such a negative effect is negligible. Basically, since optimistic simulation is performed, the construction of HM's is possible. It is because the causality error caused by the delay of message transfer during the construction of HM's can be recovered by the rollback mechanism.

4 Hierarchically Grouped Messages

4.1 Specification of HM

For the explanation of specifying HM's, Figure 3 illustrates the construction of an HM for transferring 64 data from SW to HW. First, each message represents an event (or simultaneous events) on the address/data buses or control signals such as we_b (write enable bar). The transfer of each datum is specified as a group of messages as shown in Figure 3. The transfer of 64 data is specified as a group of groups of messages, each group of which represents the transfer of a datum. As such, higher level groups of messages are constructed by grouping lower level messages (or groups of messages) in a hierarchical way.

Each HM has an (or a set of) address range(s) associated with the data belonging to the HM. In Figure 3, the HM transferring 64 data has an address range from `0x80` to `0xbc`. The designer can also specify an address range to construct an HM for the purpose of performance optimization.

4.2 Construction of HM during Simulation

During cosimulation, each simulator monitors the values on the address bus and starts to construct an HM by detecting the start address value (e.g. `0x80` in Figure 3) of the address range of the HM. During the construction of the HM, output messages are not sent to their receiving simulator. Instead, they are stored in the output message queue. If the simulator detects the end address of the address range (e.g. `0xbc` in Figure 3), then the simulator creates a physical message with the unsent messages in the output message queue and sends it to the receiving simulator. We refer to the time period between the start time and the end time of the construction of an HM as an *HM construction period*.

From the implementational viewpoint, an HM is an array of messages. From the viewpoint of the receiving simulator which reads each incoming message one by one from the Internet communication channel, there is no difference between hierarchically grouped messages and separately sent messages. The construction of an HM requires proper modifications in the cases that (1) interrupt is allowed during the construction of an HM, (2) an HM is constructed during the data dependent execution, and (3) a synchronous simulator in hybrid cosimulation constructs an HM.

4.3 Handling Interrupts

Depending on whether interrupt is allowed during communication between SW and HW, we classify the hierarchically grouped message into two types: *interruptible* HM and *non-interruptible* HM. We define an interruptible HM as follows.

Definition 1 *If the execution of SW can be interrupted while SW is constructing (or processing messages belonging to) an HM, the HM is defined to be an interruptible HM.*

For example, while SW reads 64 data from HW, the execution of SW can be interrupted by a timer interrupt to the SW processor unless the interrupt is masked. For the non-interruptible HM, the execution of SW is guaranteed to be continued during the construction or reception of the HM. For the interruptible HM, the simulator sends a *partial* HM in the cases described below. By a partial HM, we mean an HM which has been constructed until some time point before the end address of the HM is reached.

For the interruptible HM, the simulator sends a partial HM in the following two cases.

Case 1 *While the HW simulator is constructing an interruptible HM, HW sends an interrupt to SW.*

In this case, since SW execution will be interrupted by the interrupt sent by HW, the transfer of the interruptible HM is not guaranteed to continue. Thus, the HW simulator stops constructing the HM and sends the partial HM to the SW simulator.

Case 2 *During the HM construction period of an interruptible HM, the SW simulator processes a message containing an interrupt event.*

In this case, since SW execution is interrupted by the interrupt event, the SW simulator sends the partial HM to the HW simulator.

4.4 Sending a Partial HM in Data Dependent Execution

To avoid large delay caused by the data dependent executions such as data dependent loops during the construction of an HM, the simulator sends the partial HM if the delay exceeds a given timeout value ($T_{timeout}$). That is, if LVT$-T_{HM_start} > T_{timeout}$, then the simulator sends the partial HM. T_{HM_start} represents the local virtual time when the simulator starts to construct the HM. The designer can set a timeout value $T_{timeout}$. If $T_{timeout}$ = 0, then HM concept is not used.

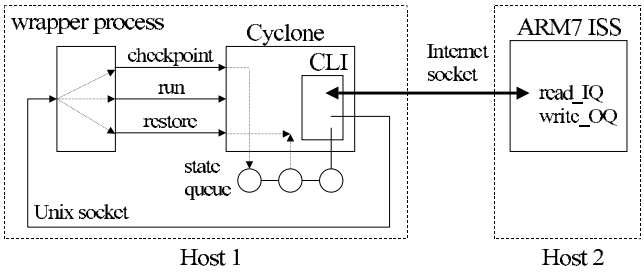

Figure 4: Optimistic distributed cosimulation using ARM7 ISS and Synopsys Cyclone.

4.5 Construction of HM in Hybrid Cosimulation

As explained in Section 2, in hybrid cosimulation the simulator stops its simulation when it sends a message to another simulator or after the time window **W** or **W'** elapses. However, in applying HM concept to hybrid cosimulation, the simulator does not stop its simulation during the construction of an HM. Therefore, it may continue the simulation beyond **W** or **W'**. After the construction of the HM, the simulator sends the HM to another simulator, stops its simulation, and waits for messages from the other simulator.

Basically, since HM concept is applied to optimistic simulation, only the optimistic simulator can construct HM's. For the non-interruptible HM, however, the synchronous simulator can also construct HM's in hybrid cosimulation since SW execution is guaranteed to be continued during the construction of the non-interruptible HM.

4.6 Calculation of GVT during the Construction of HM

In optimistic distributed cosimulation, when a simulator calculates GVT, it sends a request to the other simulators to obtain information for calculating GVT. When a simulator acknowledges to the request, it sends to the requesting simulator the minimum value of its LVT and the timestamps of unprocessed messages in its input message queue. When the simulator acknowledges to the request, if it is constructing an HM, then it sends to the requesting simulator the minimum value of its LVT, the timestamps of unprocessed input messages, and the timestamps of unsent output messages.

5 Experiments

We performed geographically distributed timed cosimulation for two examples : an H.263 decoder [7] and a JPEG encoder [8]. For the H.263 decoder, 3 frames of an image called Carphone (QCIF: 176x144 pixels) are decoded and for the JPEG encoder, a 116x96 image is encoded. For the HW parts of the examples, Discrete Cosine Transformation (DCT) and Inverse DCT functions are implemented. The other parts of the examples are implemented in SW.

We construct HM's for transferring 64 data from SW (HW) to HW (SW). In our implementation, a single original message has 44 bytes information. For transferring one datum from SW (HW) to HW (SW), four messages are transferred from the SW simulator to the HW simulator. In the case of transferring one datum from HW to SW, a single message is transferred from the HW simulator to the SW simulator together with four messages transferred from the SW simulator to the HW simulator. Thus, an HM from SW to HW contains 11,264 (=44x4x64) bytes and an HM from HW to SW contains 2,816 (=44x64) bytes in total.

We use an ARM7 instruction set simulator (ISS) having optimistic simulation features for SW simulation [9]. For optimistic HW simulation, we use a commercial cycle-based simulator, Synopsys Cyclone utilizing its checkpoint and restore functions [10]. Optimistic simulation library functions [5] are linked with ARM7 ISS and Cyclone, respectively. We also use a HW emulator [11] (based on Xilinx XC4085), which does not provide optimistic simulation features.

Table 1: Cosimulation run-times for the H.263 decoder.

N_{hop}	Opt. Dist. (sec)		Hybrid (sec)	
	w/o HM	w/ HM	w/o HM	w/ HM
3	3,727	2,436	4,579	406
12	5,900	4,200	74,577	1,457

Table 2: Cosimulation run-times for the JPEG encoder.

N_{hop}	Opt. Dist. (sec)		Hybrid (sec)	
	w/o HM	w/ HM	w/o HM	w/ HM
3	629	523	617	110
12	1,266	879	24,118	371

We use the number of *hops* N_{hop} to denote the number of Internet connections in a geographically distributed cosimulation environment.[3] To experiment the effect of communication overhead via Internet, we performed cosimulation in two different geographically distributed cosimulation environments (N_{hop} = 3 or 12). Especially, the case of N_{hop} = 12 is a connection between a workstation (or a PC) at Seoul Nat'l Univ. in Korea and a workstation at Virginia Tech. in the U.S.

For optimistic distributed cosimulation, we run ARM7 ISS and Cyclone on two remotely located simulation hosts (two SUN UltraSparc I's, 143 MHz). Figure 4 shows a simplified view of our optimistic distributed cosimulation. For the case of Synopsys Cyclone, we use C Language Interface (CLI) to link our optimistic simulation library functions with Cyclone. We also use a wrapper (a Unix process) to issue simulation commands (run, checkpoint, and restore as shown in Figure 4) to Cyclone. For hybrid cosimulation, we run ARM7 ISS (i.e. the optimistic simulator) on a workstation and the HW emulator (i.e. the synchronous simulator) on a PC (Pentium II, 300 MHz, Win98).

First, we ran uni-processor synchronous cosimulation of two examples using ARM7 ISS and Cyclone on an UltraSparc I workstation and obtained 5,816 sec (for H.263) and 1,418 sec (for JPEG) for the run-times. Table 1 and 2 show cosimulation run-times of two geographically distributed cosimulation environments. Compared to the run-times of uni-processor synchronous cosimulation, the performance improvement of optimistic distributed cosimulation (w/o HM) comes mainly from the reduction of simulator synchronization overhead rather than the benefit from parallel simulation. Applying HM concept to optimistic distributed cosimulation, we can obtain 1.53 and 1.40 times (1.20 and 1.44 times) performance improvement for the H.263 example (for the JPEG example) in the two cases of N_{hop}.

Table 3 shows the reduction of the numbers of rollbacks by applying HM concept to optimistic distributed cosimulation. Figure 5 shows the histograms of the numbers of rollbacks in the case of optimistic distributed cosimulation of the H.263 example (when N_{hop} = 12). In Figure 5, the number of short rollbacks is dramatically reduced by applying HM concept, while that of long rollbacks slightly increases due to the delay of message transfer caused by the

[3] In this paper, N_{hop} is defined to be the number of Internet routers (including gateways) plus one.

Table 3: The numbers of rollbacks in optimistic distributed cosimulation.

N_{hop}	H.263		JPEG	
	w/o HM	w/ HM	w/o HM	w/ HM
3	7,721	1,775	1,710	380
12	8,697	2,130	1,815	431

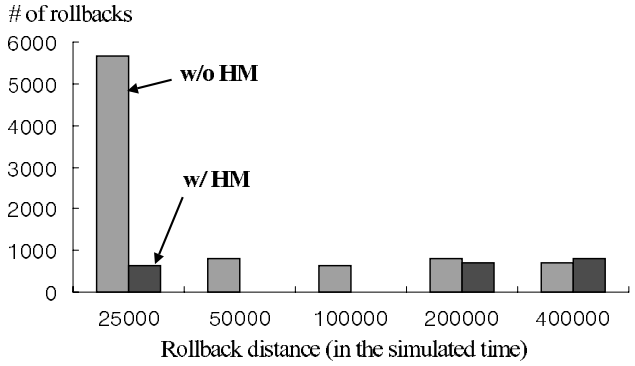

Figure 5: Histograms on rollback statistics (# of rollbacks v.s. rollback distance) in the H.263 decoder example.

construction of HM.

In Table 1 and 2, compared to the run-times of uni-processor synchronous cosimulation, the performance improvement of hybrid cosimulation (w/o HM) is mainly from the reduction of HW simulation run-time by using a HW emulator instead of the cycle-based simulator. As shown in Table 4, by applying HM concept to hybrid cosimulation, the numbers of physical messages and rollbacks are reduced down to 0.78% and 4.22% for the H.263 example, respectively (0.84% and 4.87% for the JPEG example). Such reduction of the numbers of physical messages and rollbacks gives 11.28 times (for H.263) and 5.61 times (for JPEG) performance improvement (when N_{hop} = 3) in Table 1 and 2. In hybrid cosimulation [1], since the increase of communication overhead does not change rollback behavior, Table 4 gives a single number for each type of cosimulation.

In Table 1 and 2, as the communication overhead represented by N_{hop} increases, the run-time of hybrid cosimulation without HM concept gets increased steeply due to the large numbers of physical messages and rollbacks, while HM concept gives much slower increase of run-time.

In Table 1 and 2, HM concept gives better performance improvement in hybrid cosimulation than in optimistic distributed cosimulation. The reason is as follows. In hybrid cosimulation without HM concept, two simulators synchronize at least at every message transfer as described in Section 2. For the simulator to start, it should wait to receive a message (null message or a message having an event) from other simulator(s). On the contrary, in optimistic distributed cosimulation, simulators do not stop to wait for messages. Thus, the reduction of the number of physical messages in the case of hybrid cosimulation has stronger effect on the reduction of the number of simulator synchronization, i.e. the reduction of simulator synchronization overhead including rollback overhead.

6 Conclusion

In this paper, we present hierarchically grouped message concept to reduce the simulator synchronization overhead in geographically distributed timed cosimulation. We obtained significant performance improvement by applying HM concept to geographically distributed cosimulation environments. Our experiments show that HM concept enables geographically distributed timed cosimulation to be applied in practical situations.

Currently, we are integrating hybrid and optimistic distributed cosimulation together with HM concept into an existing system design framework. Our future work includes developing efficient synchronization methods in hybrid distributed cosimulation environments where software simulators, hardware simulators, and analog simulators co-exist.

Table 4: The numbers of physical messages (No. MSG) and rollbacks (No. RB) in hybrid cosimulation.

	H.263		JPEG	
	w/o HM	w/ HM	w/o HM	w/ HM
No. MSG	684,262	5,325	163,880	1,375
No. RB	41,091	1,735	10,374	505

Acknowledgement

We would like to thank Prof. Dong S. Ha in the Dept. of ECE, Virginia Tech. for his help with the experimental setup.

References

[1] S. Yoo and K. Choi, "Synchronization Overhead Reduction in Timed Cosimulation", *Proc. IEEE International High Level Design Validation and Test Workshop*, pp. 157–164, Nov. 1997.

[2] K. Hines and G. Borriello, "Selective Focus as a Means of Improving Geographically Distributed Embedded System Co-simulation", *Proc. Eighth IEEE International Workshop on Rapid System Prototyping*, pp. 58–62, June 1997.

[3] K. Hines and G. Borriello, "A Geographically Distributed Framework for Embedded System Design and Validation", *Proc. Design Automat. Conf.*, pp. 140–145, June 1998.

[4] H. Rajaei, R. Ayani, and L. Thorelli, "The Local Time Warp Approach To Parallel Simulation", *Proc. 7th Workshop on Parallel and Distributed Simulation*, pp. 119–126, 1993.

[5] S. Yoo and K. Choi, "Optimistic Distributed Timed Cosimulation Based on Thread Simulation Model", *Proc. Int. Workshop on Hardware-Software Codesign*, pp. 71–75, Mar. 1998.

[6] D. Jaggar, *Advanced RISC Machines Architectural Reference Manual*, Prentice Hall, July 1996.

[7] Telenor, *Telenor's H.263 Software*, http://www.nta.no/brukere/DVC/h263_software/.

[8] Portable Video Research Group, *PVRG-JPEG CODEC*, ftp://havefun.stanford.edu/pub/jpeg/JPEGv1.2.1.tar.Z.

[9] W. Jang, D. Lim, and S. Yoo, *ARM7 Instruction Set Simulator*, http://poppy.snu.ac.kr/Codesign/ARMISS/.

[10] Synopsys, Inc., *Synopsys Simulation Homepage*, http://www.synopsys.com/products/simulation/simulation.html.

[11] D. Lim, K. Rha, and S. Yoo, *Design Automation Lab. Prototyping Board*, http://poppy.snu.ac.kr/Codesign/DAL_P98A/.