Performance Improvement of Geographically Distributed Cosimulation by Hierarchically Grouped Messages

Sungjoo Yoo, Kiyoung Choi, and Dong Sam Ha, *Senior Member, IEEE*

***Abstract*—To improve the performance of geographically distributed cosimulation, we propose a concept called hierarchically grouped message. The concept improves cosimulation performance, preserving the cosimulation accuracy, by hierarchically grouping messages transferred between simulators in a short period of simulated time into a single physical message, thereby reducing the number of physical messages. Applying the concept to hybrid and optimistic cosimulation, we can reduce the number of rollbacks as well as the communication overhead accompanying the message transfer. Experimental results show the efficiency of the proposed method for practical examples in an internationally distributed cosimulation environment.**

***Index Terms*—Geographically distributed cosimulation, hardware/software cosimulation, optimistic simulation, performance, rollback, simulation.**

I. Introduction

RECENTLY, a new cosimulation concept called *geographically distributed cosimulation* has been drawing more and more attention in connection with intellectual property (IP)-based design using the Internet [2], [3] or globally distributed design [4], [5]. In such cosimulation environments, designers can simulate a system that consists of remotely located IP blocks without requiring local copies of the IP blocks or subsystem designs that are being developed by their colleagues across continents. IP providers and electronic design automation vendors also gain the benefits of allowing their IP blocks and proprietary tools (e.g., high-performance hardware emulators) to be accessed remotely while protecting their IP rights and tool licenses.

A. Performance Issue in Geographically Distributed Cosimulation

In applying geographically distributed cosimulation to such Internet-based design environments, we face a significant problem in terms of cosimulation performance: high communication overhead over the Internet. To make geographically distributed cosimulation practically useful, efficient performance optimization methods should be developed that reduce the high communication overhead. Since this communication overhead is caused by the transfer of messages between simulators, reducing the number of messages transferred between simulators is crucial.

There are two types of messages transferred among simulators involved in geographically distributed cosimulation: event-carrying messages and null messages.[1] In cosimulation of communication-intensive systems such as H.263 or wireless code division multiple access (CDMA) systems, the communication overhead of transferring event-carrying messages over the Internet can dominate cosimulation run-times. Since hardware (HW) and software (SW) simulators should synchronize with each other (via slow communication over the Internet) at every system clock tick to detect the occurrence of interrupt, the communication overhead of transferring null messages over the Internet can be prohibitively large, especially when *interrupt* is used as one of communication protocols in the system being designed.

Thus, the performance optimization methods of geographically distributed cosimulation should reduce both numbers of event-carrying messages and null messages simultaneously.

B. Previous Work

Distributed cosimulation has been extensively researched [6]–[8]. However, since geographically distributed cosimulation is a relatively new area, little research has been aimed at improving the performance of geographically distributed cosimulation.

As an effective method of improving the performance of geographically distributed cosimulation, a concept called *selective focus* [9]–[11], [2] has been proposed. It allows designers to change the abstraction levels of communication models dynamically during cosimulation. Thus, the designers can trade off between cosimulation performance and accuracy.

In [3], a geographically distributed simulation of IP-based designs called *virtual simulation* is presented. In the work, to improve the simulation performance, a group of simulation inputs (called patterns) to an IP block are buffered and then sent to a remotely located host, which then estimates the power consumption of the IP block in a batch process manner.

Manuscript received August 13, 2000. This work was supported by ETRI, Korea. A preliminary version of this work was presented as a regular paper in *Proc. Int. Workshop on Hardware-Software Codesign*, May 1999, pp. 100–104. This work was supported in part by ETRI, Korea.

S. Yoo is with SLS Group, TIMA/INPG, Grenoble 38031, France, on leave from Seoul National University, Seoul 151–742 Korea

K. Choi and D. S. Ha are with Seoul National University, Seoul 151–742 Korea.

Publisher Item Identifier S 1063-8210(00)09506-8.

[1]Null messages are used for simulator synchronization only; i.e., to notify the local time of the message sending simulator.

Hybrid cosimulation proposed in [12] and [13] reduces the number of null messages in distributed cosimulation environments where optimistic simulators and conservative simulators[2] coexist. Since optimistic simulation has an advantage in such a case that synchronization overhead is dominant [14], *optimistic cosimulation* [15] can reduce the communication overhead by reducing the number of null messages. For the reduction of optimistic simulation overhead such as state-saving overhead, thread-based state-saving methods have been proposed in [15].

In the societies of distributed simulation and parallel programming, there have been a few studies on improving simulation performance through *message aggregation*. In [16], to reduce the communication overhead of transferring messages, transformation techniques such as loop unrolling are applied to parallel programs to aggregate messages into a less number of physical messages. In [17], to reduce rollbacks caused by frequent exchanges of messages between optimistic simulators, message aggregation is performed on an *aggregation window* basis. Messages that occur inside the window (which slides on simulated time) are aggregated into a physical message.

C. Contribution of the Paper

In this paper, to reduce the number of event-carrying messages, we present a new concept called *hierarchically grouped message* (HGM). Basically, the HGM concept utilizes the fact that transmitting one large message is faster than transmitting multiple small-sized messages one by one. Correspondingly, the communication overhead over the Internet does not strictly depend on the sizes of messages being transferred, but rather strongly depends on the number of physical messages transferred.

The HGM concept proposed in this paper differs from the *message aggregation* concept in the following aspects. In [17], the aggregation window is set to an interval of simulated time ignoring the semantics of messages. By contrast, in the HGM concept, the designer or an automated tool specifies HGMs using the semantics of messages—higher level information on messages transferred between SW and HW simulators—thereby obtaining more efficient cosimulation. In addition, the HGM concept is applied to hybrid cosimulation where conservative simulators and optimistic simulators coexist and considers cosimulation-specific characteristics such as handling interrupt.

Previous approaches on geographically distributed cosimulation present limitations. Some of these limitations require the designer to trade off between simulation accuracy and performance [9]–[11], [2]; other limitations prevent optimization methods from being applied to timed cosimulation [3]. Compared to the previous approaches, the proposed HGM concept has the advantage of improving the performance of geographically distributed timed cosimulation while preserving the timing accuracy. Furthermore, by integrating the HGM concept with hybrid and optimistic cosimulation, we can further improve the performance of geographically distributed timed cosimulation by reducing both the number of null messages as well as that of event-carrying messages.

[2]In this paper, we call a simulator that does not perform optimistic simulation a *conservative* simulator.

D. Organization of this Paper

This paper is organized as follows. In Section II, we give preliminaries, including terminology and basic assumptions used throughout this paper. In Section III, we describe the message grouping concept in hybrid and optimistic cosimulation. In Section IV, we address the issues of managing the HGM in hybrid and optimistic cosimulation. We give experimental results in Section V and conclude this paper in Section VI.

II. Preliminaries

In this section, we explain the terminology, assumptions, and the concepts of hybrid and optimistic cosimulation used in this paper.

A. Terminology

1) *Message:* A timestamped event. In this paper, if there is no confusion between null messages and event-carrying messages, we use the term *messages* to denote event-carrying messages.
2) *Optimistic/conservative simulators:* Optimistic simulators can perform rollback while conservative simulators cannot.
3) *Local virtual time (LVT) and global virtual time (GVT):* Each simulator has its own local time, called local virtual time. Global virtual time is the minimum of timestamps of in-transit messages[3] and local virtual times of simulators.
4) *Straggler message:* A message that has a timestamp earlier than the LVT of the receiving simulator.
5) *Rollback:* When an optimistic simulator receives a straggler message, it rolls back; that is, it restores a state whose timestamp is not later than the timestamp of the straggler message.
6) *Antimessage:* After rollback, to cancel a previously sent output message that has a timestamp later than the LVT, the simulator sends an antimessage to the simulator that received the message to be canceled.
7) *Aggressive/lazy cancellation policies:* In the aggressive cancellation policy, the simulator sends antimessages just after rollback. In the lazy cancellation policy, the simulator defers sending antimessages until its LVT again reaches the timestamp of the output message to be canceled. Then, after comparing the contents of the previously sent output message with those of a new output message to be sent at that time, it determines whether or not to send the antimessage.

B. Assumptions

Fig. 1 shows an example of communication interface between SW and HW from [18]. The dashed boxes enclosing the SW and HW parts denote corresponding simulators. The SW and HW simulators exchange messages carrying events across the

[3]In-transit messages are messages that are in the communication channels among simulators, or not processed yet in input message queues. In our implementation, the Internet communication channel works as a first-in first-out queue.

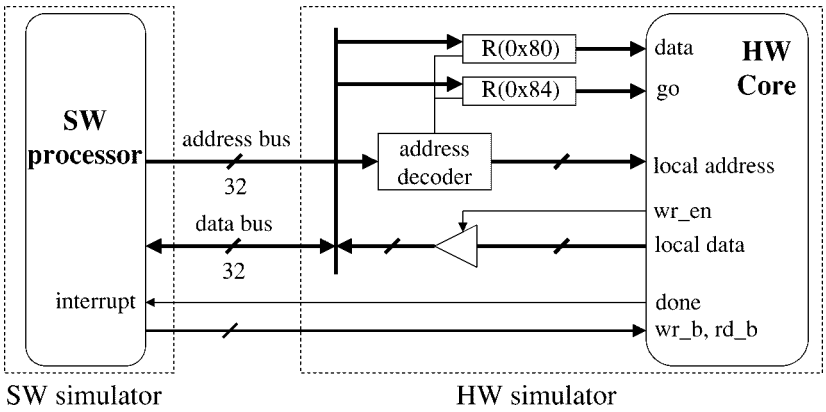

Fig. 1. An example of communication interface between SW and HW.

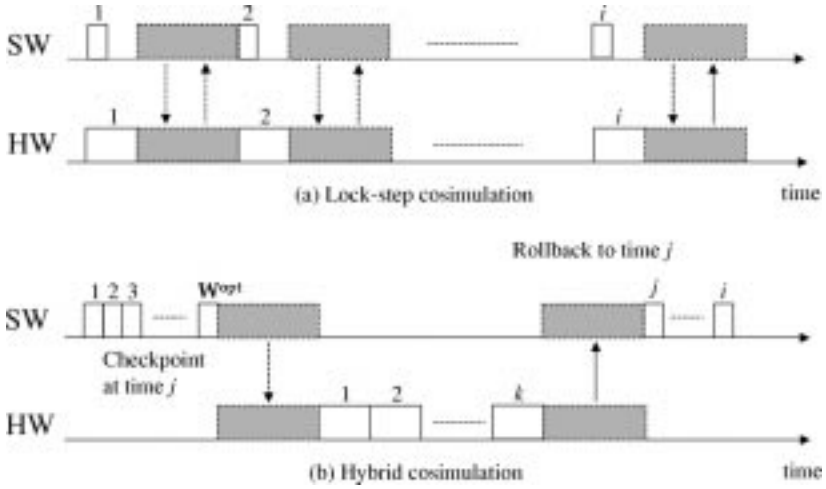

Fig. 2. Hybrid cosimulation reduces synchronization overhead caused by null messages.

boundary between the SW and the HW through the address/data buses, control pins such as wr_b and rd_b, and the interrupt pin.[4] To simplify explanation, in this paper, we assume that data transfer between SW and HW is performed using memory-mapped I/O and interrupts. However, the HGM concept can be easily extended to the cases in which other types of HW-SW communication such as port-mapped I/O are used. For simplicity's sake, only a few number of communication signals are considered, as shown in Fig. 1. We also assume that the cosimulation is performed with two processes: one for the SW simulator and the other for the HW simulator, as shown in Fig. 1. The concept can be also applied to cosimulation with multiple processes [13].

C. Hybrid and Optimistic Cosimulation

In this section, we briefly describe hybrid cosimulation and optimistic cosimulation.

1) Hybrid Cosimulation: To explain hybrid cosimulation, we first introduce lock-step cosimulation. Fig. 2(a) shows an example of lock-step cosimulation. In lock-step cosimulation, simulators advance their local times by one system clock period in a lock-step manner and synchronize with each other at every system clock tick exchanging (null) messages to detect the occurrence of events (e.g., interrupt) to be exchanged. In the figure, white rectangles and the numbers on them represent simulation workloads and the corresponding local times in the simulators, respectively. Shaded rectangles represent *simulator synchronization overhead* in terms of runtime, which is caused by the transfer of null messages and event-carrying messages. Dashed arrows between simulators represent null messages, while solid arrows represent event-carrying messages. As shown in Fig. 2(a), in lock-step cosimulation, synchronization overhead can dominate total cosimulation runtime.

To reduce such overhead, we perform hybrid cosimulation. In hybrid cosimulation, optimistic and conservative simulators are involved. Fig. 2(b) shows an example of hybrid cosimulation scenario. Assume that the solid arrow in this figure represents a message carrying an interrupt event from HW to SW. We assume that the SW simulator performs optimistic simulation and the HW simulator does not. First, the optimistic simulator runs its simulation for an adaptively controlled time window $\mathbf{W^{opt}}$, saving its states at checkpoints (at time j in this example) in preparation for a potential rollback. It stops the simulation at time W^{opt} and sends a null message (the dashed arrow in the figure) to the conservative simulator (HW simulator). Then the optimistic simulator waits for a message to come from the conservative simulator.

[4]In our experiments, a single message consists of timestamp, source/destination simulator ID's, message ID, message type, address/data bus values, control pin values, interrupt pin values, etc.

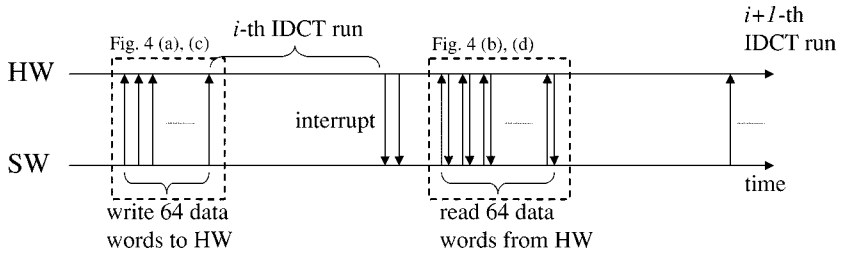

Fig. 3. An example of message communication: an H.263 decoder system.

After receiving a message from the optimistic simulator, the conservative simulator starts to run until the time point W^{opt} where the optimistic simulation has stopped. The conservative simulation may stop earlier at time k, $k < W^{opt}$ as shown in the figure, if the conservative simulator comes to send an event-carrying message (in this example, an interrupt event) to the optimistic simulator at that time. In this case, since the timestamp of the message sent to the optimistic simulator (time k) is earlier than the time point where the optimistic simulator has stopped (time W^{opt}), the optimistic simulator rolls back to the checkpoint at time j, which is not later than the timestamp of the straggler message (time k). However, if there is no event to be transferred from the conservative simulator to the optimistic simulator, then the conservative simulator stops at the time point where the optimistic simulator has stopped (time W^{opt}) and sends a null message to the optimistic simulator.

After updating the size of W^{opt}, the optimistic simulator starts to run until the new window elapses. The cosimulation continues running 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^{opt} elapses. For more details of hybrid cosimulation, refer to [12] and [13].

2) Optimistic Cosimulation: When the simulators involved in the cosimulation are all optimistic simulators, we perform optimistic cosimulation. Each optimistic simulator has its own LVT and manages its state queue and input/output message queues. Each optimistic simulator works as follows. It looks up the input message queue to find an input message having a timestamp equal to LVT, processes the message (if any), and advances its LVT. If there is any straggler input message, then the optimistic simulator rolls back its LVT according to the timestamp of the straggler message.

To constrain the memory usage for state saving in the simulation host, GVT is calculated. States and messages having timestamps earlier than GVT can be removed from the state queue and the input/output message queues.[5] For the details of optimistic cosimulation, refer to [15].

Even when there are conservative simulators involved in the cosimulation, if there is also more than one optimistic simulator involved, we can still apply the optimistic cosimulation concept to the coordination among the optimistic simulators. However, we apply the hybrid cosimulation concept to the coordination between optimistic simulators and conservative simulators [13].

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

III. Message Grouping in Hybrid and Optimistic Cosimulation

Fig. 3 shows an example of message transfer between the SW and HW simulators in the cosimulation of an H.263 decoder system [19]. We assume that SW writes 64 data words to the HW, which implements the inverse discrete cosine transformation (IDCT) function. On the completion of IDCT, an interrupt is sent to the SW to signal the completion. The execution time of the IDCT function is not assumed to be fixed (the minimum and maximum bounds may be given). After receiving the interrupt, the SW reads 64 data words from the HW as a result of the IDCT function. In the processor where the SW runs, the write and read operation can be performed by executing memory store and load instructions (e.g., STR, STM, LDR, and LDM instructions in an ARM7 processor [20]). To write each of the data words, the SW sends the address value and data value to the HW; for example, in Fig. 4(a), `address(0)` and `data(0)`. To receive each of the data words, after the SW sends the address value to the HW, the HW sends the data word corresponding to the received address value to the SW.

A. Message Grouping in Hybrid Cosimulation

In this section, we assume that the SW simulator performs optimistic simulation and the HW simulator does not. For the output message cancellation in optimistic simulation, we adopt lazy cancellation policy. Fig. 4(a) shows a hybrid cosimulation scenario where 64 data words are transferred from SW to HW. Fig. 4(b) shows the case of data transfer from HW to SW. These two cases correspond to the two dashed boxes in Fig. 3. In Fig. 4, numbers beside arrows represent the execution order in each cosimulation scenario. In the figure, thick arrows represent the execution of each simulator.

In Fig. 4(a), the optimistic simulator (SW simulator) runs first (arrow numbered 1) and stops when it sends a message (containing an address value, e.g., *address(0)*) to the conservative simulator (arrow numbered 2). After receiving the message, the conservative simulator runs until the time point where the optimistic simulator has stopped (arrow 3); it then sends a null message (arrow 4) to notify the optimistic simulator that the conservative simulator has run up to the destined time point. If the cosimulation scenario in Fig. 4(a) is performed in a geographically distributed cosimulation environment, the network communication overhead caused by the large number of small-sized messages[6] can yield serious performance degradation.

[6]In our implementation, the size of a single ungrouped message is 44 bytes.

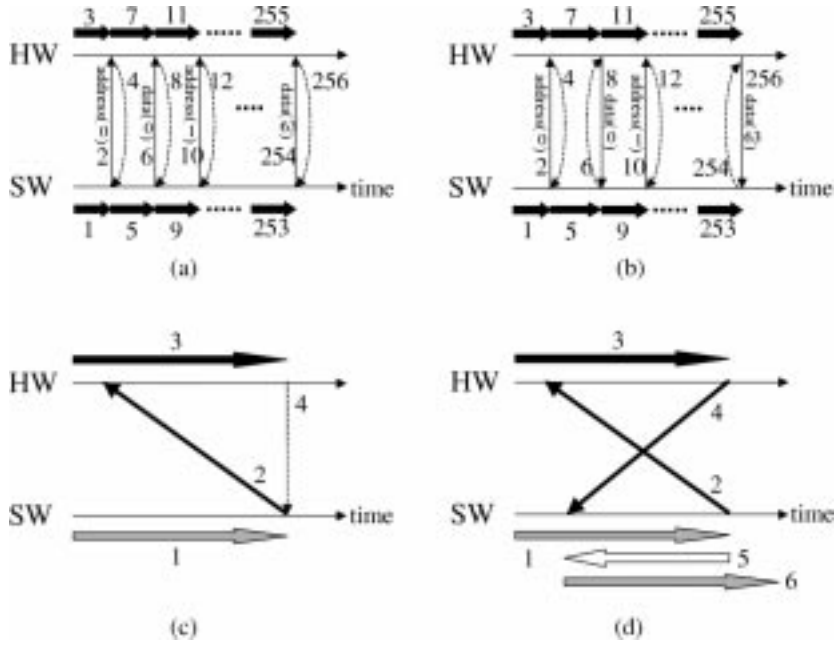

Fig. 4. Grouping messages to transfer multiple data words.

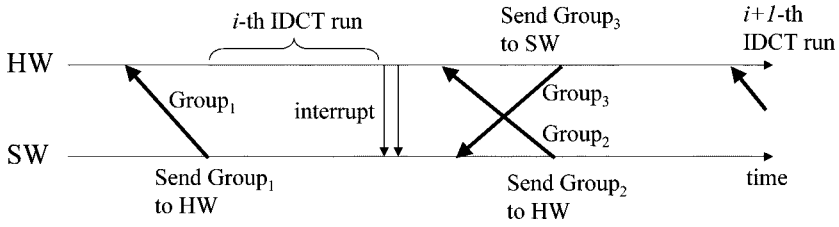

Fig. 5. Hybrid cosimulation of the H.263 decoder system using groups of messages.

To resolve such a problem caused by the transfer of a large number of small-sized messages, we can transfer multiple data words at a time by grouping them as shown in Fig. 4(c). In this example, we put 128 messages (64 messages for address values and 64 messages for data values) transferred from SW to HW into a group of message (thick arrow 2). In constructing a group of messages during cosimulation, we do not raise the abstraction levels of original messages or events, but keep them unchanged. Thus, the timing accuracy of cosimulation can be maintained after the grouping process. We simply delay sending messages belonging to a group until all the messages belonging to the group are ready to be sent. When the messages are ready, they are grouped into a physical message and sent to the receiving simulator (HW simulator). In Fig. 4(c), the group of messages is represented by the thick arrow numbered 2 between the SW and HW simulators. This figure illustrates that the concept of message grouping can give significant reduction in the number of physical messages.

Fig. 4(d) shows a hybrid cosimulation scenario using message grouping for the case of Fig. 4(b). Since the SW reads 64 data words from the HW, the SW simulator sends to the HW simulator a group of messages (thick arrow 2), which contain 64 messages for address values. The HW simulator sends a group of messages (thick arrow 4) containing 64 messages for data values. In Fig. 4(d), the group of messages (thick arrow 4) sent by the HW simulator contains messages whose timestamps are earlier than the LVT of the SW simulator. Thus, the SW simulator should roll back. In Fig. 4(d), the white arrow numbered 5 represents rollback in SW simulation. Since we adopted lazy cancellation policy for output message cancellation, after rollback the optimistic simulator does not have to cancel the output messages already sent in the group of messages (thick arrow 2).

In the case of the given example, result data items of IDCT operation do not change the addresses (thick arrow 2) sent before. Thus, lazy cancellation prevents resending the messages containing the addresses. In the case that the received data items change the addresses and the optimistic SW simulator cancels the previously sent output messages to send new messages containing changed addresses (i.e., there is *circular dependency* among messages exchanged between two simulators), since the new messages are straggler messages to the conservative simulator, causality error can occur on the side of the conservative simulator. Thus, in hybrid cosimulation, the HGM concept should not be applied to the message group with circular dependency among messages.

By applying the concept of message grouping to the example in Fig. 3, we can reduce the number of physical messages as shown in Fig. 5. Group_1 in the figure represents messages transferred from the SW simulator to the HW simulator while the SW writes 64 data words to the HW. Group_2 and Group_3 are transferred while the SW reads 64 data words from the HW.

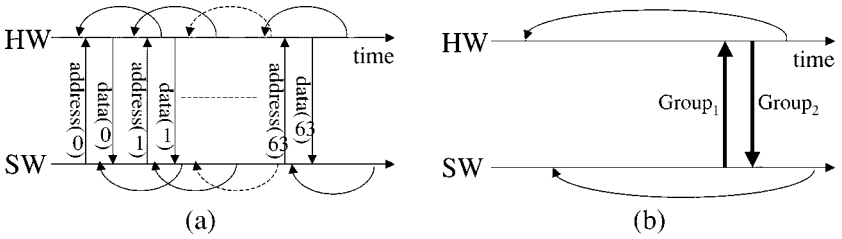

Fig. 6. Reduction of rollbacks by grouping messages.

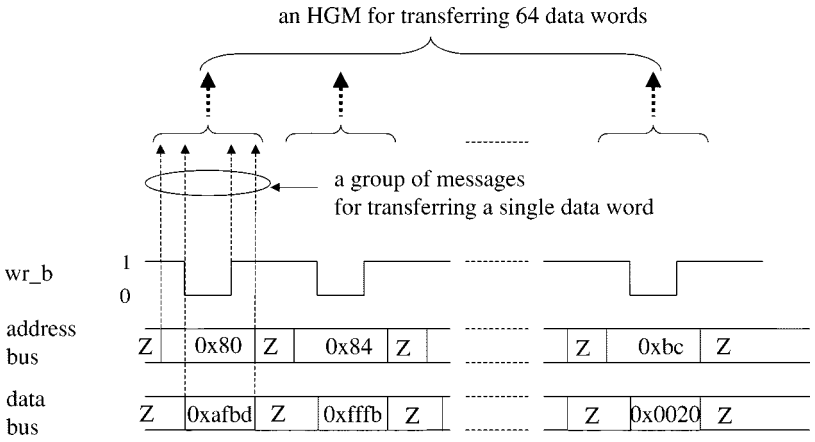

Fig. 7. An example of constructing an HGM.

B. Message Grouping in Optimistic Cosimulation

The concept of message grouping also has the advantage of reducing the number of rollbacks in optimistic cosimulation. Fig. 6 illustrates an example of message communication between SW and HW simulators in optimistic cosimulation, where we assume that both SW and HW simulators perform optimistic simulation. We also assume a case that the SW reads 64 data words from the HW, which corresponds to the case in Fig. 4(b).

In memory load (or store) instructions that are performed for the SW to read (write) 64 data words from (to) the HW, the time gap between the event on the address bus [e.g., `address(0)`] and the associated event on the data bus [e.g., `data(0)`] 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 data word requested by the SW arrives at the SW simulator, the SW simulator (e.g., one that has millions of cycles/second 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). The leftward arcs in Fig. 6 represent rollbacks caused by such straggler messages.

As shown in Fig. 6(a), optimistic cosimulation suffers from excessive rollbacks when the message transfer between simulators is performed intensively in a short period of simulated time. By grouping messages, we can reduce such excessive rollbacks as shown in Fig. 6(b). In this example, we group 64 messages transferred from the SW (HW) to the HW (SW) into a single physical message Group_1 (Group_2). Since only two physical messages are transferred, in Fig. 6(b), rollback occurs only twice in total.

Note that, basically, the concept of grouping messages is applicable since we are assuming optimistic simulation. While the delay of message transfers can cause causality errors in the message-sending simulator when the message-receiving simulator sends straggler messages to the message-sending simulator, in optimistic simulation, however, the causality errors can be recovered by the rollback mechanism.

IV. Management of Hierarchically Grouped Messages in Hybrid and Optimistic Cosimulation

The grouping of messages is done hierarchically. Fig. 7 illustrates an example of constructing such a hierarchically grouped message (HGM) when the SW writes 64 data words to the HW. The addresses of the 64 data words are assumed to range from `0x80` to `0xbc`.[7] Each of the events on the address/data buses or control signals such as wr_b generates a message transferred between the SW and HW simulators. For the construction of an HGM, first we build a subgroup with messages carrying events on the address/data buses and control signals for transferring a single data word as shown in Fig. 7. Then we group the subgroups into a single HGM. As such, higher level groups of messages are constructed by grouping lower level messages or subgroups in a hierarchical way.

A. Specification of HGM Based on Regular Expression

An HGM is specified manually by the designers or an automated tool.[8] In cases where the data transfer between SW and

[7]The address values are assigned in the communication synthesis step before timed cosimulation is performed.

[8]Currently, we are investigating the possibility of grouping messages automatically so that the cosimulation performance is optimized.

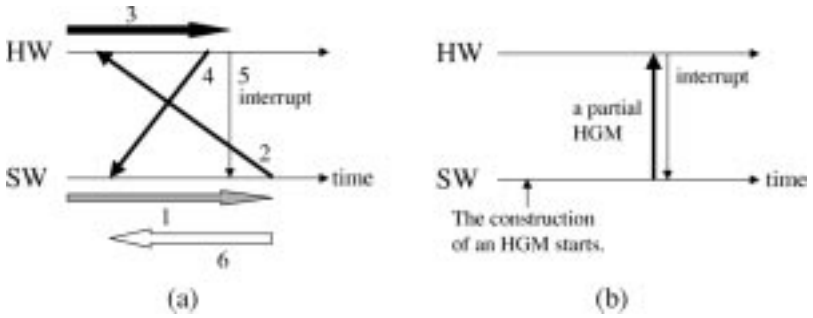

Fig. 8. Examples of sending partial HGMs.

HW is performed using memory mapped I/O, each HGM has a *sequence of address values* associated with it. Thus, we specify the HGM based on the associated sequence of address values.

As a formal method to specify an HGM, we use *regular expression* [21]. For example, in Fig. 7, the HGM transferring 64 data words has a sequence of address values ranging from `0x80` to `0xbc`. Thus, it can be specified by a regular expression `(0x80)(0x84)(0x88)···(0xbc)`. For another example, we recall Fig. 1 and assume that the SW writes 64 data words to the HW using the same address value of `0x80`. The HW core reads the data words one by one from the same memory location synchronous with the write operation. After transferring the whole set of 64 data words, the SW sends a job initiation signal called *go* whose address value is assumed to be `0x84` to the HW to initiate the execution of the HW core. In this case, the designer specifies the HGM using a regular expression `(0x80)64(0x84)`.[9]

At the beginning of the construction of an HGM, there can be multiple candidate regular expressions that partially match the address sequence being monitored. The simulator finishes constructing the HGM and sends it to the receiving simulator when one of the candidate regular expressions yields a perfect match with the address sequence. If it is found that no regular expression yields a perfect match, then the simulator sends a *partial HGM*. By a partial HGM, we mean a set of messages that belong to an HGM and have been collected up to some time point where the regular expression of the HGM is not satisfied yet. In Section IV-C, the cases in which the partial HGM should be sent will be described in detail.

B. Construction of HGM During Cosimulation

During cosimulation, each simulator monitors the values on the address bus and starts to construct an HGM by detecting the start address value (e.g., `0x80` in Fig. 7) of the regular expression of the HGM. During the construction of the HGM, output messages are not sent to their receiving simulator. Instead, they are stored in the output message queue. If the sequence of address values satisfies the regular expression of the HGM (for the example of Fig. 7, if the simulator detects the end address `0xbc`), 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 HGM as an *HGM construction period*.

From the implementational viewpoint, an HGM is an array of messages. From the viewpoint of the receiving simulator that reads each incoming message one by one from the Internet communication channel, there is no difference between messages sent as a group and individually sent messages.

C. Sending a Partial HGM

The HGM construction process requires some modification in the cases where 1) interrupt occurs during the construction of an HGM and 2) an HGM is constructed during the data dependent execution.

1) Handling Interrupt: The simulator can send a partial HGM on the occurrence of interrupt. Consider hybrid cosimulation, for example, where the SW reads 64 data words from the HW. Assume that the execution of SW can be interrupted (e.g., by a timer interrupt to the SW processor) during the read operation. Fig. 8(a) illustrates a case which is actually similar to the case shown in Fig. 4(d). The difference is that, in Fig. 8(a), an interrupt is sent to the SW while the HW simulator is constructing an HGM which is supposed to carry 64 data words going to the SW. In this case, since the SW execution will be interrupted by the interrupt sent by the HW, the transfer of the whole set of 64 data words is not guaranteed. Thus, the HW simulator stops constructing the HGM and sends the partial HGM [thick arrow 4 in Fig. 8(a)] as well as the interrupt message (arrow 5) to the SW simulator. Then the SW simulator rolls back (white arrow 6) due to the partial HGM (arrow 4).

Fig. 8(b) shows the same case in optimistic cosimulation. In the figure, while the SW simulator is constructing an HGM that will contain messages of 64 address values, an interrupt is sent to SW. In this case, since the SW execution is interrupted by the interrupt event, the SW simulator stops constructing the HGM and sends the partial HGM (possibly after rollback), e.g., the messages collected up to the time point where the SW receives the interrupt, to the HW simulator.

2) Construction of HGM in Data-Dependent Execution: To avoid a lengthy delay caused by data-dependent executions (e.g., data-dependent loops) during the construction of an HGM, the simulator can send a partial HGM if the delay exceeds a timeout value (T_{timeout}) set by the designer. Since T_{timeout} is used only to control the construction of HGM, T_{timeout} is considered only during the HGM construction period. In

[9]To the other cases of HW-SW communication protocols, regular expression can be easily applied.

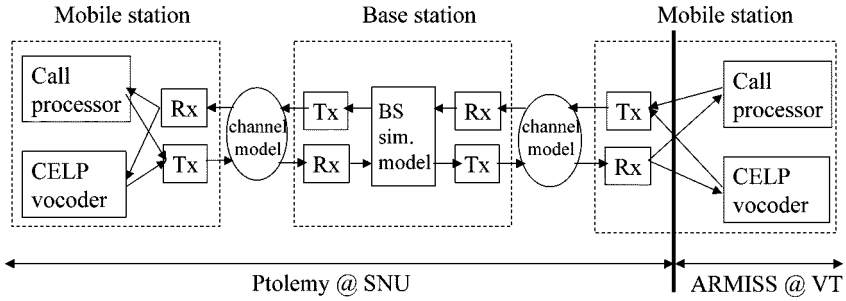

Fig. 9. Geographically distributed cosimulation of a wireless CDMA cellular phone system.

optimistic cosimulation, if LVT $-$ $\text{LVT}_{\text{HGM\_start}} > T_{\text{timeout}}$, where $\text{LVT}_{\text{HGM\_start}}$ represents the LVT when the simulator starts to construct the HGM, then the simulator sends a partial HGM. In hybrid cosimulation, optimistic simulation runs for the time interval whose size is the minimum of T_{timeout} and W^{opt}. Thus, if LVT $-$ $\text{LVT}_{\text{HGM\_start}} > \min(T_{\text{timeout}}, \text{W}^{\text{opt}})$, then the simulator sends a partial HGM. If T_{timeout} is set to zero, then the HGM concept is not used.

D. Construction of HGM in Hybrid Cosimulation

As explained in Section II-C1, in hybrid cosimulation the simulator stops its simulation when it sends a message to the other simulator or after the time window W^{opt} elapses. However, when we apply the HGM concept to hybrid cosimulation, the optimistic SW simulator does not stop its simulation during the construction of an HGM. It may continue the simulation beyond W^{opt}. After the construction of the HGM, it stops the simulation, sends the HGM to the other simulator, and waits for a message to come from the other simulator.

Basically, only the optimistic simulator can construct HGMs since potential causality errors caused by the delay of message transfer must be recovered by the rollback mechanism. However, in the case that the SW simulator is an optimistic one, the conservative HW simulator can also construct the HGM.

E. Calculation of GVT During the Construction of HGM

In optimistic cosimulation, every simulator calculates GVT autonomously when the calculation is required. When a simulator needs to calculate GVT, it sends to other simulators a request for information required to calculate GVT. Each simulator receiving the request acknowledges by sending the minimum among its LVT and the timestamps of unprocessed messages in its input message queue. If a simulator receives such a request while the simulator is constructing an HGM, then it sends to the requesting simulator the minimum among its LVT, the timestamps of unprocessed input messages, and the timestamps of unsent output messages.

V. EXPERIMENTS

A. Examples

We perform geographically distributed cosimulation for three examples: a wireless CDMA cellular phone system [22], an H.263 decoder [19], and a JPEG encoder [23].

Fig. 9 shows the CDMA system based on the IS-95 specification [24]. The system consists of two mobile stations (MSs), a base station (BS), and air channel models. The MS and BS communicate with each other on a 20-ms frame basis. Call processor and code excited linear prediction vocoder in the MS generate the frame data and send them to the transmitter (Tx) of the CDMA modem in the MS. The Tx in the MS sends the frame data via the air channel model to the BS. The BS receives the frame data through its receiver (Rx), processes them for such operations as call initialization, conversation, registration, etc., then sends the frame data to the MS or another MS. In our experiment, we run cosimulation for 60-frame data.

For the H.263 decoder, three frames of a video image called Carphone (QCIF: 176 $\times$ 144 pixels) are decoded, and for the JPEG encoder, a 116 $\times$ 96 image is encoded. For the H.263 and JPEG examples, discrete cosine transformation (DCT) and inverse DCT functions are implemented in HW. The other parts of the two examples are implemented in SW.

B. Simulator Configurations

Table I shows the simulator configurations of our experiments. For the CDMA system, we use Ptolemy [25] (as the conservative simulator) and an ARM7 instruction set simulator (ISS) having optimistic simulation features [26] in hybrid cosimulation. Since Ptolemy does not perform optimistic simulation, we omit optimistic cosimulation for the CDMA system.

For hybrid cosimulation of the H.263 and JPEG examples, we use the ARM7 ISS for optimistic SW simulation and a HW emulator [27] based on Xilinx XC4085. The HW emulator is connected to the network through a PC and does not perform optimistic simulation but takes the role of a conservative simulator. In optimistic cosimulation, we use a commercial cycle-based simulator, Synopsys Cyclone, for optimistic HW simulation utilizing its checkpoint and restore functions [28]. For optimistic cosimulation, we use optimistic simulation library functions [15], which are linked with the ARM7 ISS and Cyclone. Fig. 10 shows a simplified view of our optimistic cosimulation. For the case of Synopsys Cyclone, we use C language interface (CLI) to link the optimistic simulation library functions with Cyclone. We also use a wrapper process (a Unix process) to issue simulation commands such as run, checkpoint, and restore to Cyclone, as shown in Fig. 10. We run the ARM7 ISS, Synopsys Cyclone, and Ptolemy on UltraSparc I workstations with the clock speed of 143 MHz.

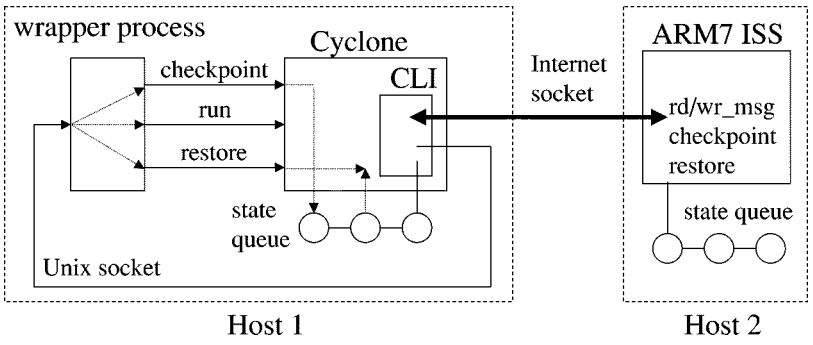

Fig. 10. Optimistic cosimulation using ARM7 ISS and Synopsys Cyclone.

In Table I, we use the number of *hops* $N_{\rm hop}$ to denote the number of Internet connections in a geographically distributed cosimulation environment.[10] To investigate the effect of communication overhead via Internet, we experimented with two different geographically distributed cosimulation environments: one with $N_{\rm hop} = 3$ and the other with $N_{\rm hop} = 12$. The one with $N_{\rm hop} = 3$ uses two workstations and a PC connected by the campus network (10-Mbps ATM LAN) at Seoul National Univ (SNU) in Korea. The case of $N_{\rm hop} = 12$ uses a connection between a workstation and a PC at SNU and a workstation at Virginia Institute of Technology (VT) in the United States. In the case of the CDMA system, as shown in Fig. 9, we simulate the call processor and vocoder part of the MS on the ARM7 ISS located at VT and the other part of the CDMA system on Ptolemy located at SNU in Korea.

TABLE I
SIMULATOR CONFIGURATIONS

		hybrid cosim.		opt. cosim.	
		opt.	cons.	opt. (SW)	opt. (HW)
CDMA		ARM7 ISS	Ptolemy	-	-
H.263, JPEG		ARM7 ISS	emulator	ARM7 ISS	Cyclone
$N_{\rm hop}$	3	SNU	SNU	SNU	SNU
	12	VT	SNU	VT	SNU

TABLE II
RUN-TIMES (SECONDS) OF HYBRID COSIMULATION

$N_{\rm hop}$	CDMA		H.263		JPEG	
	w/o HGM	w/ HGM	w/o HGM	w/ HGM	w/o HGM	w/ HGM
3	1,248	1,060	4,579	406	617	110
12	46,323	2,361	74,577	1,457	24,118	371

C. Specification of HGMs

According to the IS-95 specification [24], there are four rates of frame data specified for the *reverse traffic channel* from the MS and the BS to which we apply the HGM concept in our experiments.[11] We construct HGMs for transferring one frame data(16, 40, 80, or 172 data words depending on the rate) between the MS call processor/vocoder and the MS modem (Tx and Rx). For the H.263 and JPEG examples, we construct HGMs for transferring 64 data between SW and HW. In our implementation, a single nongrouped message has 44 bytes of information. For SW to write one data word to HW (e.g., STR instruction in ARM7 processor), four messages are transferred from the SW simulator to the HW simulator.[12] In the case that SW reads one data word from HW (e.g., LDR instruction in ARM7 processor), 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, in the CDMA example, an HGM from the MS call processor/vocoder to the MS modem contains 2816 $(=44 \times 4 \times 16)$ to 30 272 $(=44 \times 4 \times 172)$ bytes and an HGM from the MS modem to the MS call processor/vocoder contains 704 $(=44 \times 16)$ to 7568 $(=44 \times 172)$ bytes. In the H.263 and JPEG examples, an HGM from SW to HW contains 11 264 $(=44 \times 4 \times 64)$ bytes and an HGM from HW to SW contains 2816 $(=44 \times 64)$ bytes.

D. Experiments for Hybrid Cosimulation

Table II gives run-times[13] of hybrid cosimulation for three examples. Compared with the cases where the HGM concept is not applied, by applying the HGM concept, we can obtain 1.18 times (CDMA), 11.28 times (H.263), and 5.61 times (JPEG) performance improvement when $N_{\rm hop} = 3$. As the communication overhead increases—i.e., as $N_{\rm hop}$ increases—the run-times of hybrid cosimulation without the HGM concept increase steeply. However, by applying the HGM concept to the hybrid cosimulation, cosimulation run-times increase much slower so that we can obtain higher performance improvement up to 19.62 times (CDMA), 51.16 times (H.263), and 65.01 times (JPEG) in the case of $N_{\rm hop} = 12$, as shown in Table II. In our experiments of hybrid cosimulation, when $N_{\rm hop} = 12$, the network traffic condition causes up to 30% variation (maximum–minimum) of cosimulation run-time.[14]

As shown in Table III, by applying the HGM concept to hybrid cosimulation, the numbers of physical messages are reduced down to 0.35% (CDMA), 0.78% (H.263), and 0.84%

[10]In this paper, $N_{\rm hop}$ is defined as the number of Internet routers (including gateways) plus one. $N_{\rm hop}$ is obtained by running a program called *traceroute*.

[11]In our experiments, we do not apply the HGM concept to such other channels as access channel, pilot channel, sync channel, page channel, and forward traffic channel.

[12]In our current implementation of ARM7 ISS, we model LDR/STR instructions with four different states. In each state, the ARM7 ISS sends a message to the HW simulator.

[13]In our experiments, we obtained average run-times by running cosimulation three times.

[14]Actually, simulation run-times vary depending on the network condition. Even in a single day, the network condition fluctuates frequently. Thus, the variation of simulation runtimes can be more than 30% for the cosimulation environments with different network configurations.

TABLE III
NUMBERS OF PHYSICAL MESSAGES IN HYBRID COSIMULATION

CDMA		H.263		JPEG	
w/o HGM	w/ HGM	w/o HGM	w/ HGM	w/o HGM	w/ HGM
124,589	442	684,262	5,325	163,880	1,375

TABLE IV
RUN-TIMES (SEC.) OF OPTIMISTIC COSIMULATION

N_{hop}	H.263		JPEG	
	w/o HGM	w/ HGM	w/o HGM	w/ HGM
3	3,727	2,436	629	523
12	5,900	4,200	1,266	879

TABLE V
NUMBERS OF PHYSICAL MESSAGES IN OPTIMISTIC COSIMULATION

N_{hop}	H.263		JPEG	
	w/o HGM	w/ HGM	w/o HGM	w/ HGM
3	42,244	2,190	13,922	650
12	42,587	2,691	14,081	775

(JPEG). Such drastic reduction is the very source of significant performance improvement shown in Table II. The effects of such reduction in the numbers of physical messages become more evident in the case of $N_{\text{hop}} = 12$, where much higher network communication overhead consumes the majority of simulation run-time.

E. Experiments for Optimistic Cosimulation

As shown in Table IV, by applying the HGM concept to optimistic 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}. Note that the cycle-based HW simulator is used in optimistic cosimulation for the HW part simulation while the HW emulator is used in hybrid cosimulation. Thus, run-times of optimistic cosimulation in Table IV are longer than those of corresponding cases in hybrid cosimulation in Table II. Table V shows that by applying the HGM concept, the numbers of physical messages are reduced down to 5.18% (H.263) and 4.67% (JPEG). Table VI shows the reduction in the numbers of rollbacks by applying the HGM concept to optimistic cosimulation. We obtain more than four times reduction in the numbers of rollbacks for the two examples.

Fig. 11 shows the histograms of the number of rollbacks when $N_{\text{hop}} = 12$. In the figure, *rollback distance* represents the amount of simulated time canceled by rollback. The figure shows that the numbers of short rollbacks are dramatically reduced by applying the HGM concept, while those of long rollbacks slightly increase. The increase in the numbers of long rollbacks is due to the delayed message transfer for the construction of the whole HGM. In our experiments, however, such an increase does not noticeably degrade optimistic cosimulation performance.

Comparing the data in Tables II and IV, we see that the HGM concept is much more effective in hybrid cosimulation than in optimistic cosimulation. The reason is as follows. In hybrid

TABLE VI
NUMBERS OF ROLLBACKS IN OPTIMISTIC COSIMULATION

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

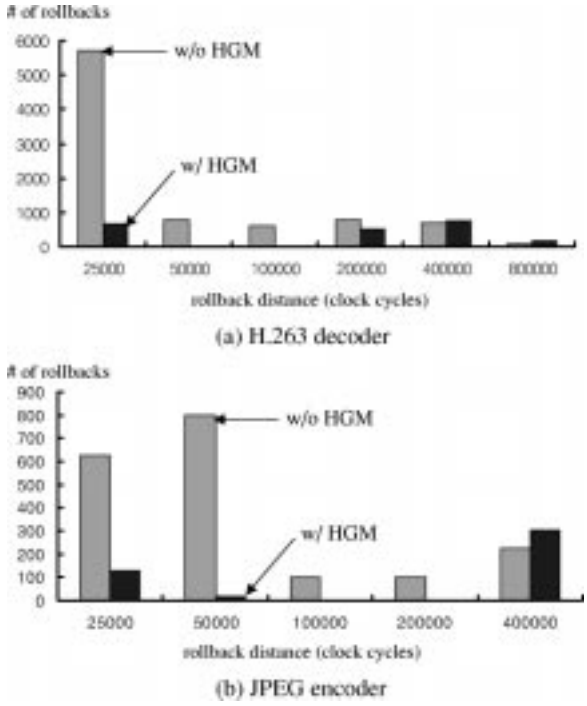

Fig. 11. Histograms on rollback statistics (number of rollbacks versus rollback distance) in the H.263 decoder and JPEG encoder examples.

cosimulation without the HGM concept, two simulators synchronize at least once at every message transfer, as described in Section II-C1. For one simulator to continue its simulation, it should wait to receive a message (a null message or an event carrying message) from the other simulator. Therefore, grouping messages saves much time otherwise consumed for such synchronization. On the contrary, in optimistic cosimulation, simulators do not stop to wait for messages. Therefore the amount of time consumed for the synchronization is small even when the HGM concept is not used.

VI. CONCLUSION

In this paper, we propose the concept of hierarchically grouped message to improve the performance of geographically distributed cosimulation by reducing the number of physical messages transferred between simulators. We have obtained significant performance improvement by applying the proposed concept to geographically distributed cosimulation of practical examples even in an internationally distributed cosimulation environment. Our experiments show that the HGM concept enables geographically distributed cosimulation to be applied to practical situations.

Currently, we are integrating hybrid and optimistic cosimulation together with the HGM concept into an existing system design framework. We are also investigating the problem of automating the message grouping process, which is currently done

manually. Our future work includes developing efficient synchronization methods in hybrid distributed cosimulation environments where software simulators, hardware simulators, and analog circuit simulators coexist.

REFERENCES

[1] S. Yoo and K. Choi, "Optimizing geographically distributed timed cosimulation by hierarchically grouped messages," in *Proc. Int. Workshop Hardware-Software Codesign*, May 1999, pp. 100–104.

[2] K. Hines and G. Borriello, "A geographically distributed framework for embedded system design and validation," in *Proc. Design Automation Conf.*, June 1998, pp. 140–145.

[3] M. Dalpasso, A. Bogliolo, and L. Benini, "Virtual simulation of distributed IP-based designs," in *Proc. Design Automation Conf.*, June 1999, pp. 50–55.

[4] R. Goering. (1999, Jan.) Global chip design raises promises and challenges. *EE Times* [Online] http://www.eetimes.com/story/OEG19 990 111S0016

[5] H. Lavana, A. Khetawat, F. Brglez, and K. Kozminski, "Executable workflows: A paradigm for collaborative design on the internet," in *Proc. Design Automation Conf.*, June 1997, pp. 553–558.

[6] D. E. Thomas and S. L. Coumeri, "A simulation environment for hardware-software codesign," in *Proc. Int. Conf. Computer Design*, Oct. 1995, pp. 58–63.

[7] A. Ghosh, M. Bershteyn, R. Casley, C. Chien, A. Jain, M. Lipsie, D. Tarrodaychik, and O. Yamamoto, "A hardware-software co-simulator for embedded system design and debugging," in *Proc. Asia South Pacific Design Automation Conf.*, 1995.

[8] C. Valderrama, F. Nacabal, P. Paulin, and A. Jerraya, "Automatic VHDL-C interface generation for distributed cosimulation: Application to large design examples," *Design Automat. Embedded Syst.*, vol. 3, pp. 199–217, Mar. 1998.

[9] K. Hines and G. Borriello, "Optimizing communication in embedded system co-simulation," in *Proc. Int. Workshop Hardware-Software Codesign*, Mar. 1997, pp. 121–125.

[10] ——, "Selective focus as a means of improving geographically distributed embedded system co-simulation," in *Proc. 8th IEEE Int. Workshop Rapid System Prototyping*, June 1997, pp. 58–62.

[11] ——, "Dynamic communication models in embedded system co-simulation," in *Proc. Design Automation Conf.*, June 1997, pp. 395–400.

[12] S. Yoo and K. Choi, "Synchronization overhead reduction in timed cosimulation," in *Proc. IEEE Int. High Level Design Validation and Test Workshop*, Nov. 1997, pp. 157–164.

[13] ——, "Optimizing timed cosimulation by hybrid synchronization," in Design Automation for Embedded Systems. Norwell, MA: Kluwer Academic , June 2000, vol. 5, pp. 129–152, to be published.

[14] H. Rajaei, R. Ayani, and L. Thorelli, "The local time warp approach to parallel simulation," in *Proc. 7th Workshop Parallel and Distributed Simulation*, 1993, pp. 119–126.

[15] S. Yoo and K. Choi, "Optimistic distributed timed cosimulation based on thread simulation model," in *Proc. Int. Workshop Hardware-Software Codesign*, Mar. 1998, pp. 71–75.

[16] W. Gropp and E. Lusk. Tuning MPI applications for peak performance. [Online] http://www-unix.mcs.anl.gov/mpi/tutorial/perf/index.html

[17] M. Chetlur and N. Abu-Ghazaleh, "Optimizing communication in time-warp simulators," in *Proc. 12th Workshop Parallel and Distributed Simulation*, May 1998, pp. 64–71.

[18] Y. Kim, K. Kim, Y. Shin, T. Ahn, and K. Choi, "An integrated cosimulation environment for heterogeneous systems prototyping," *Design Automat. Embedded Syst.*, vol. 3, no. 2/3, pp. 163–186, Mar. 1998.

[19] Telenor. Telenor's H.263 software. [Online] http://www.nta.no/brukere/DVC/h263_software/

[20] D. Jaggar, *Advanced RISC Machines Architectural Reference Manual*. Englewood Cliffs, NJ: Prentice-Hall, July 1996.

[21] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools*. Reading, MA: Addison-Wesley, 1988.

[22] Qualcomm, Inc., *CDMA System Engineering Training Handbook*, 1993.

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

[24] TIA/EIA-95A, "Mobile station-base station compatibility standard for dual-mode wideband spread spectrum cellular systems,", 1995.

[25] J. T. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, "Ptolemy: A framework for simulating and prototyping heterogeneous systems," *Int. J. Comput. Sim. (Special Issue on Simulation Software Development)*, vol. 4, pp. 155–182, Apr. 1994.

[26] W. Jang, D. Lim, and S. Yoo. ARM7 instruction set simulator. [Online] http://poppy.snu.ac.kr/Codesign/ARMISS/

[27] D. Lim, K. Na, and S. Yoo. Design automation lab. prototyping board. [Online] http://poppy.snu.ac.kr/Codesign/DAL_P98A/

[28] Synopsys, Inc. Cyclone VHDL reference manual. [Online] Synopsys Online Documentation, v1998.08

Sungjoo Yoo received the B.S. degree in electronics engineering and the M.S. and Ph.D. degrees in electrical engineering from Seoul National University, Korea, in 1992, 1995, and 2000, respectively.

His research interests include hardware–software cosimulation, performance estimation in system-level design, low-power system design, and reconfigurable system design.

Kiyoung Choi received the B.S. degree in electronics engineering from Seoul National University, Korea, in 1978 and the M.S. degree in electrical and electronics engineering from Korea Advanced Institute of Science and Technology, Korea, in 1980. He received the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, in 1989.

From 1978 to 1983, he was with GoldStar, Inc., Korea, and from 1989 to 1991, he was with Cadence Design Systems, Inc. In 1991, he joined the Faculty of the Department of Electronics Engineering, Seoul National University, as an Assistant Professor. In 1995, he joined the Faculty of School of Electrical Engineering, Seoul National University, where he is now an Associate Professor. His primary interests are in VLSI design and various aspects of computer-aided design, including hardware–software codesign, high-level synthesis, and low-power systems design.

Dong Sam Ha (S'83–M'85–SM'97) received the B.S. degree in electrical engineering from Seoul National University, Korea, in 1974 and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of Iowa, Iowa City, in 1984 and 1986, respectively.

Since 1986, he has been a Faculty Member of the Bradley Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University, Blacksburg. Currently, he is a Professor with the department. Prior to his graduate studies, he was a Research Engineer for the Agency for Defense Development, Korea, from 1975 to 1979. While on leave from May to December of 1996, he was with the Semiconductor Research Center of Seoul National University, where he investigated built-in self-test synthesis. He was a Guest Researcher of the German National Research Center for Computer Science (GMD) near Bonn, Germany, in the summer of 1994. He, along with his students, has developed four CAD tools for digital circuit testing. The source code for these four tools has been distributed to more than 130 universities and research institutions worldwide. The tools have been used for various research and teaching purposes at numerous universities. His research interests include low-power VLSI design for wireless communications, low-power wireless video system design, low-power analog and mixed-signal VLSI design, design for testability, and built-in self-test.