Optimized Timed Hardware Software Cosimulation without Roll-back*

Wonyong Sung Soonhoi Ha

Department of Computer Engineering
Seoul National University
Seoul, Korea 151-742
e-mail: {yong,sha}@iris.snu.ac.kr

Abstract— An optimized hardware software cosimulation method based on the backplane approach is presented in this paper. To enhance the performance of cosimulation, efforts are focused on reducing control packets between simulators as well as concurrent execution of simulators without roll-back.

I. Introduction

Cosimulation has gained extensive research focuses but with diverse approaches depending on their different emphasis on conflicting goals such as cosimulation speed, accuracy, flexibility, and so on[1]. While our proposed approach is based on a cosimulation environment that puts most emphasis on the flexibility, this paper addresses how to improve the cosimulation speed. To make a flexible cosimulation environment, we use a heterogeneous approach rather than a unified one. In the heterogeneous approach, the software simulator and the hardware simulator are separate processes running concurrently and cooperatively. We propose a backplane approach, in which a new simulator has only to define the interface to the backplane, leaving existent simulators unchanged[2].

As system complexity grows, cosimulation speed becomes a great concern. Research efforts on cosimulation speedup include to use hardware accelerator, to change cosimulation models across levels of abstractions, and to accelerate cosimulation speed by reducing the cosimulation overheads. To reduce the cosimulation overhead, one may use the optimistic approach in which a component simulator rolls back when it receives a past event from the other simulator. This approach assumes that the component simulator supports rollback mechanism that is usually not the case. In this paper, we propose another approach to optimize the conservative timed cosimulation.

II. Optimized Cosimulation

Basic Conservative Timed Cosimulation: For timed cosimulation the backplane has a priority queue sorted by the timestamp in the packet. Since all the communication messages are passed through the backplane, the backplane knows the global status of the cosimulation. After the global clock of the backplane advances to the global next event's time, the backplane sends the message with timestamp to the client simulator. The client simulator executes until its local clock reaches the global current time and sends the response packet. On receiving the response packet, the backplane schedules this response packet to its event queue. Then, the backplane sends message to another simulator, if there is a message to the simulator at the current time. When there is no more event pending at the global time and the client process is blocked, it informs to the backplane the local time of the next earliest event scheduled after the global time. Then, the backplane schedules a dummy event to the client simulator, which just awakes the client simulator to process the events at the new global time. Since not all the event driven simulators allow the user to get the next event time, the local next time is assumed a unit time increment from the current local time in the basic mode. While this scheme always guarantees timing correctness, it will be very inefficient especially when the event interval of the client simulator is much shorter than that of the backplane.

Optimization Mode 1 : If the client simulator has an API to get the next event time, the client simulator requests to the backplane to schedule itself at the next event time. For example, Synopsys VSS VHDL simulator provides **getNextEventTime()** in the CLI library. It reduces the synchronization points drastically specially when the time unit of the client simulator is much smaller than the event interval.

Optimized Mode 2 : In mode 2, we allow the local clock to be ahead of the global clock if the client simulator is confirmed not to receive any past event. If there is only one client simulator, the backplane sends the global next event time, which becomes the stop time of the client simulator, to the client simulator along with input messages. Then, the client simulator can proceed until its local clock reaches the given stop time or it generates a response mes-

*This work was supported by Ministry of Education through Interuniversity Semiconductor Research Center(ISRC-96-E-2103) in Seoul National University

TABLE I
RUNTIME PERFORMANCE OF COSIMULATION IN VARIOUS TIMED COSIMULATION MODE

Mode	Total(msec.)	BP	HW	SW	IPC
Basic	2,105,881	13.7%	4.2%	0.1%	82%
Mode1	473,277	39.8%	0.9%	0.4%	58.9%
Mode2	308,643	75.1%	1%	0.7%	23.2%

TABLE II
NUMBER OF CONTROL/DATA PACKETS(UNIT : PACKET)

Mode	Total	Data	Control
Basic	667,047	3,842	663,205
Mode1	24,805	3,842	20,963
Mode2	21,755	3,842	17,913
Mode3	29,801	10,535	19,266

sage to the backplane. If it generates a response message before the stop time, the message becomes the next event in the backplane.

Optimization mode 3 : Up to optimized mode 2, the backplane waits until the client simulator responds after it sends input messages. Therefore, no parallelism is exploited in the distributed cosimulation. In mode 3, we allow to send the simultaneous messages to multiple client simulators at the current global time. The global clock of the backplane, however, can not advance until it receives response packets from the client simulators. We run client simulators concurrently in the distributed environment. The more optimization is possible when the partitioned graph is feed-forward or acyclic. We topologically sort the partitioned graph that corresponds to a client simulator each. Then, the ancestor simulator can be ahead of the descendant simulator. Therefore, the backplane keeps track of event time for each arc, not for the whole event queue. The backplane allows a past event to the global queue as long as it is not a past event on the associated arc. In our experiments, only the top-most client simulator run in optimization mode 2. The other simulators run in mode 1 because the backplane is hard to guarantee that it will not receive any past event from the ancestor simulator after local clock advancement.

TABLE III
COMPARISON COSIMULATION TIME BETWEEN SERIALIZED AND PARALLEL EXECUTION IN MODE 3 (UNIT : MSEC.)

	Total	HW1	HW2	SW	IPC
Serial	934,169	7,515	4,076	2,088	411,672
Parallel	586,772	7,417	4,212	2,048	403,998

III. EXPERIMENTAL RESULTS

We define the cosimulation time into four parts: VHDL simulation time(HW), software simulation time(SW), cosimulation backplane time(BP), and interprocess communication time(IPC). After we cosimulate the QAM example with 320 loop counts, we get the profiling results presented in table I and III. Under assumption that the transmission bandwidth is 19,200bps, the 320 loop counts means 1 second of real time. The reason why the VHDL simulation time is much smaller than the total cosimulation time is that the VHDL model is very simple. The main source of low performance of cosimulation is the IPC overhead mainly due to the control packets between the backplane and the client simulator. Table II shows how many control packets are needed in each mode. The IPC time as well as the number of control packets is reduced using mode 1 and 2. The backplane parts, which includes the global event queue management, is also reduced according to the reduction of the number of control packets To experiment the optimization mode 3, we partition the hardware module into two parts. Each sub-module is simulated by separate Synopsys VSS simulator run on the different machine. Even though more computational resources are used, as shown in table 3, the QAM simulation in mode 3 shows worse performance due to the partition of the VHDL module. The other reason is that only one of the two VHDL simulators uses the optimized mode 2. Since the optimization approach in mode 3 is quite different and even the model graph is also different, comparison of the result of mode 3 with those of other modes is not so meaningful. Thus, we compare the result in case of serialized run of simulators with that in case of parallel run. Although the items are almost same, the total runtime is reduced in case of parallel run because of the runtime overlap.

IV. CONCLUSION

Based on our cosimulation backplane environment, cosimulation speedup approaches are devised and implemented. Shrinking the number of control packets and utilizing concurrent simulation reduce the cosimulation runtime while the flexibility of backplane is maintained. According to the various model topology, selective adaptation of optimized mode will be possible.

REFERENCES

[1] S. Edwards, L. Lavagno, E. A. Lee, and A. Sangiovanni-Vincentelli, "Design of Embedded Systems: Formal Methods, Validation, and Synthesis", Proceedings of the IEEE, Vol. 85, No. 3,March 1997.

[2] W. Sung and S. Ha, "Hardware Software Cosimulation Backplane with Automatic Interface Generation", ASPDAC'98, 1998.