Proceedings of the 2013 Winter Simulation Conference
R. Pasupathy, S.-H. Kim, A. Tolk, R. Hill, and M. E. Kuhl, eds.

ON-TIME DATA EXCHANGE IN FULLY-PARALLELIZED CO-SIMULATION WITH CONSERVATIVE SYNCHRONIZATION

Asim Munawar
Takeo Yoshizawa
Tatsuya Ishikawa
Shuichi Shimizu

IBM Research - Tokyo
6-52, Toyosu 5-Chome, Koto-ku
Tokyo 135-8511, JAPAN

ABSTRACT

Trade-offs between simulation speed, fidelity, compatibility, and scalability limits the use of accurate high-resolution simulators in the automotive industry. With a growing demand for fuel-efficient and environmentally friendly vehicles, the need for precise co-simulation of entire vehicle is greater than ever before. In this paper we present a technique for distributed discrete event co-simulation that exploits parallel computing and distributed simulation with an advanced synchronization technique to overcome all of these constraints. The system allows us to add new components with their own solvers to a simulation without compromising the solution accuracy or simulation speed.

1 INTRODUCTION

Virtual unit testing is a widely employed technique in the automotive industry. However, in recent years vehicles are becoming increasingly complex due to growing number of new and improved electronic, mechanical, and mechatronic technologies. To deal with these complexities in a cost-effective way, automotive makers are moving towards an era of virtual integrated testing of multiple components. Such integrated testing involves a variety of components that are each best simulated by different simulation tools. Co-Simulation (co-operative simulation) is a methodology that allows the simulation of heterogeneous components running simultaneously and exchanging information in a collaborative manner. The scalability requirements for the integrated testing of 10s or even 100s of components can be provided by using Distributed Discrete Event Simulation (DDES) techniques over parallel hardware. However, the existing technologies suffer from speed-accuracy trade-offs and therefore fail to provide sufficient fidelity for fully parallel executions. In contrast, increasing the accuracy with conventional techniques unacceptably slows the simulation speed. Ideally, the fidelity of the results for each component during the integrated simulation should be the same as the fidelity during unit testing of that component.

In this paper, we propose a DDES technique that exploits the "time shifts" that are naturally allowed between simulation units to achieve a fully parallelized co-simulation of complex systems with on-time data exchange in a conservative synchronization manner (i.e. no rollbacks are required to recover the exact data exchange points). The technique was developed for DDES in general with a focus on the automotive industry. Our contribution is a framework for co-simulation of automobile in a parallel environment with:

- No compromise of *fidelity*.
- No decrease in *simulation speed* with the addition of new components.
- 100% parallelism i.e. Simulation time is determined only by the slowest component.
- Deadlock-free simulation.

978-1-4799-2076-1/13/$31.00 ©2013 IEEE

Section 2 clarifies the problems that our work seeks to solve. The key idea of the research is discussed in Section 3. We further elaborate the details of the proposed framework in Section 4. Some empirical experiments and their results are explained in Section 5. Comparison with existing techniques appear in Section 6. Finally we conclude the paper with some directions for future work in Section 7.

2 BACKGROUND & MOTIVATION

Modern automobiles consist of increasingly complex engines, cruise controls, and powertrain systems that are typically controlled by over 60 Electronic Control Units (ECUs) (Pretschner et al. 2007). As the vehicles become smarter, the dependencies and communications among the different components are increasing. To perform accurate simulations for a modern vehicle, the most reasonable solution involves integrated simulations of the whole vehicle. A practical simulator for whole vehicle simulation should be *scalable* enough to run a high precision simulation in a reasonable amount of time. In this section, we argue that in the existing vehicle simulation techniques the scalability comes at the cost of execution speed or solution's fidelity.

Classical tools for continuous simulations construct a unified model of the whole physical system. Although the modules are logically separated for design purposes the tool does not consider these logical divisions during the actual simulation. Dependencies are resolved among different components and the execution is scheduled accordingly. Such simulation techniques are most suitable for serial execution (though limited parallelism can be achieved in some cases).

Discrete time simulation or time-driven simulation is another widely used simulation technique. In this approach each of the simulation components run the same simulation time step independently and the communication is limited to predefined synchronization points. This approach is inefficient for events dispersed irregularly over time. A co-simulation strategy derived from this technique is known as the macro-step approach. This technique is easy to fully parallelize by assuming a unit-delay on all the connections. This delay on the connections directly affects the fidelity of the solution, especially when several components are connected in a serial fashion. Deciding on the step size is also a difficult task. Smaller steps increase the accuracy of the simulations, while larger steps are required for speed and efficiency.

A more practical approach is to design a DDES-based co-simulation system where a large model is broken down into smaller sub-models called physical processes. One physical process represents a single component of the actual system (such as an engine or an ECU). Each physical process is then simulated by a Logical Process (LP) that can run independent of the other LPs. Each LP maintains its own independent simulation clock called Local Virtual Time (LVT). All DDES systems require some kind of synchronization mechanism to advanced the LVT. The synchronization techniques can be broadly divided into two categories, optimistic and conservative approaches. Comprehensive comparisons of optimistic and conservative approaches can be found in Alois and Satish (1994) and Vee and Hsu (1999).

Optimistic synchronization takes risks that might violate the causality constraint (i.e. causes have to precede their effects). This approach is based on the work presented in Jefferson and Sowizral (1982), Jefferson and Sowizral (1985), and Jefferson (1985) as time-warp mechanism. In an optimistic approach the LPs continue to process events as soon as they are received, even though this could cause some messages to be incorrectly processed out of order. If an event is received with a timestamp smaller than LVT, the simulation is rolled back to the time where the erroneous processing occurred, and the simulation is restarted so that events occur in the correct order. Events sent erroneously are cancelled by sending "anti-events". Due to frequent rollbacks the performance of an optimistic approach degrades as the number of LPs increases. This approach is impractical in some cases, such as Hardware-In-Loop (HIL) simulations. Also, depending on the simulation tool, a particular component may not support rollback.

Conservative synchronization is based on the work presented in Chandy and Misra (1979), Misra (1986) and Bryant (1984). In conservative approach events are always processed in chronological order. This handling of incoming events with exact timestamps yields precisely the same results as obtained by a centralized (sequential) execution. In order for the LPs to proceed in parallel, they must know the next

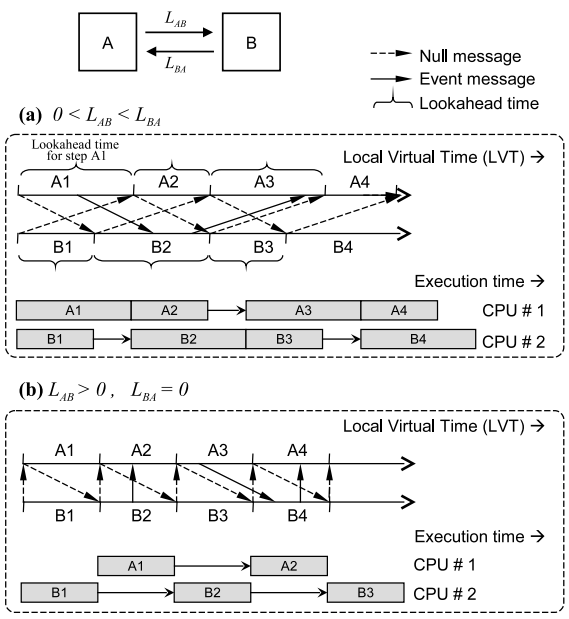

Figure 1: Serialization with a conventional conservative approach. {A1,A2,...}, {B1,B2,...} represent time steps in LP A and B respectively: (a) Partial serialization, (b) Full serialization.

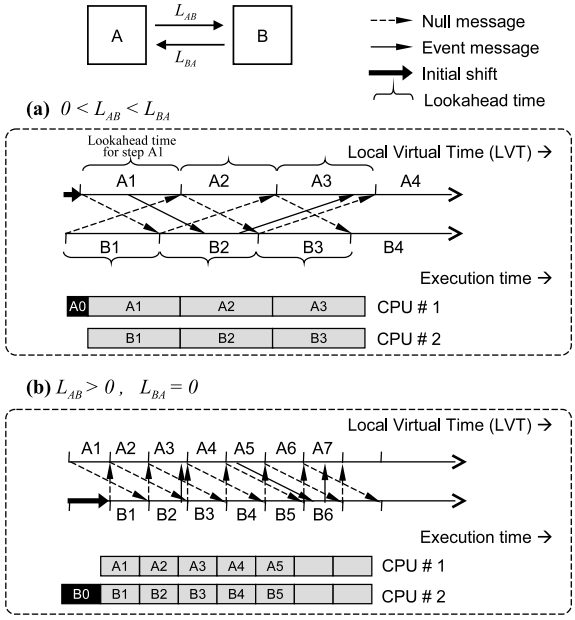

Figure 2: Full parallelism can be achieved by introducing an initial-shift in some logical processes. This figure can be compared with Figure 1 to see the advantage of the proposed approach.

lookahead time, during which the other LPs will never send any events to the LP. When an LP receives at least one event on all of its input ports, the lookahead time can be computed as the minimum timestamp among all of the received events. The LP can then proceed safely up to this time. Since some of the LPs may not receive events from all of their sources, they cannot advance, which causes a deadlock situation. To avoid deadlocks, the LPs need to send null messages to their destination LPs in addition to the event messages. However, a naively designed scheme could lead to an explosion in the number of null messages exchanged or even deadlock. The conservative approach fails to achieve 100% parallelization (Alois and Satish 1994, Vee and Hsu 1999) in simulations where two directly connected LPs have different lookahead properties (Figure 1(a)). L_{AB} in the figure represents the lookahead time between LP $A \rightarrow B$. For a lookahead value of 0 in one direction the conventional approach fails to achieve any parallelization and the simulation must proceed sequentially (refer to Figure 1(b)). When compared with an optimistic approach, a conservative approach shows better scalability for larger models (Perumalla 2007).

In this paper we extend the conservative approach so that the distributed discrete event simulation can run in a multiprocessor environment with fully parallelized execution. In addition, fidelity is never compromised in this process of increasing the simulation speed through parallel execution.

3 THE CORE CONCEPT

The key idea behind this research is to shift some of the LPs in time to achieve high fidelity on-time synchronization (i.e. the events are handled without loosing any fidelity) with full parallelization. We exploit the fact that the simulation units can run in slightly shifted time periods without violating the causality constraint. Time shift execution with null-messages enables 100% parallel execution of DDES with on-time data exchange. These shifts in time are realized by introduction of "initial time shifts" in the LPs. Initial-shifts are computed so that neighboring pairs have the same lookahead time with each other, which guarantees full parallelization with no deadlock. Initial-shift is introduced in LP *A* of Figure 2(a)

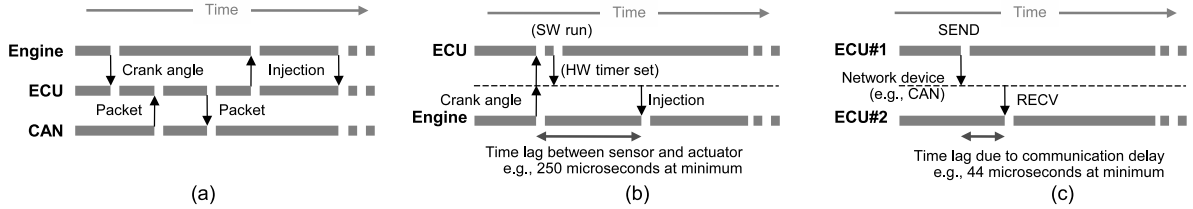

Figure 3: Time-lags can be found in many physical systems: (a) Ideal communication for synchronization between models, (b) Time-lag due to decision and action time, (c) Time-lag due to communication delay.

and in LP B of Figure 2(b). The technique never violates causality constraint and therefore falls into the category of conservative synchronization in DDES.

The proposed technique relies upon the minimum time-lag properties of the physical system to compute the lookahead time. The time-lag is a natural communication or processing delay that exists in every electrical and mechanical system. Figure 2 can be compared with Figure 1 to see how the proposed technique achieves full parallelization by introducing an initial-shift in one of the processes. L_{AB} and L_{BA} in this figure represents the time-lag from LP $A \rightarrow B$ and LP $B \rightarrow A$ respectively. We can see that even in the case of 0 time-lag full parallelization is achieved (Figure 2(b)). In contrast the conventional conservative algorithm fails to achieve any parallelism as is evident from Figure 1(b). In Figure 2(b) the time-lag $L_{BA} = 0$ and $L_{AB} > 0$. By introducing an initial-shift of $L_{AB}/2$ at LP B, we can calculate equal lookahead $(L_{AB} + L_{BA})/2$ in both directions. Equal lookahead time in both directions ensure a fully parallel execution. Note that both the LPs are running in a time shifted manner, time step $A1$ of LP A and time step $B1$ of LP B runs in parallel with B lagging behind A. Due to this time shifted execution each LP can generate an event at any point in time and the receiver LP can catch the event without compromising the fidelity. Even though the total number of null-messages exchanged in a given amount of time increases when compared to a conventional conservative approach, we now can achieve full parallelization and reduce the simulation time by using the hardware resources more effectively.

Figure 3 further illustrates the concept of time-lag. Ideal communication for synchronization between multiple components is shown in Figure 3(a). In simulation systems, we cannot predict the future rendezvous points for exact synchronization but we can determine the minimum communication delay and the minimum time period a LP takes to reply to an incoming request. For example with an ECU and an Engine, the ECU should receive state update events from the Engine with zero delay, but the Engine will not receive any actuator signals for some time after the events (250 μs [1]), because ECU takes some time to decide the actuator signals and also HW timer has non-zero interval until its alarm. Therefore, as shown in Figure 3(b) the Engine is allowed to go forward (250 μs) in time without waiting for ECU and without losing accuracy. Similarly, as shown in Figure 3(c), the CAN communication takes at least 44 μs [2] for a single data packet at 1-Mbps mode. The communication delay varies according to the size of data packets, but, it is never less than 44 μs. Therefore, each ECU is allowed to proceed 44 μs without losing any fidelity.

[1]The engine ECU is designed to consume the crank-angle pulse from the engine. A crank-angle pulse is generated at every 15° of crank-angle and is therefore generated 24 times in 1 revolution of the crank shaft. Assuming 10,000 rpm (revolutions-per-minute) as the maximum limit and 24 crank-angle pulses generated in 1 second, the frequency of pulses become $(10,000 \times 24)/60$ Hz. Therefore, the minimum time delay can be calculated to be 250 μs.

[2]Referring to Bosch CAN version 2.0 Robert Bosch GmbH (1991) specifications the CAN bit rate is up to 1Mbit/s, so a 1-bit transfer consumes at least 1μs. According to the same document, each data frame of a standard format contains at least a Start of frame(1 bit) + Identifier(11 bits) + RTR(1 bit) + Control field(6 bits) + Data field(0 bit) + CRC Sequence(15 bits) + CRC Delimiter(1 bit) + ACK Field(2 bits) + End of frame(7 bits) = 44 bits. Therefore, 44 bits x 1μs/bit = 44 μs time-lag.

4 CCSS ARCHITECTURE

We named the proposed technique Complex Control Systems Simulation (CCSS). CCSS is a set of tools and runtime packages that accelerate integrated simulations of complex control systems. In contrast to standard simulators used in the industry, CCSS makes sure that the elapsed time of integrated simulation never increases except for a small overhead even when new components are added to the system. Simulator units can run in parallel on top of multiple computational resources with very little overhead for communication and initial scheduling. Also, the simulation's fidelity is never degraded as long as minimum time-lags of data exchange communication are defined between logical processes.

4.1 Distributed Framework of CCSS

CCSS allows co-simulation of multiple components that run simultaneously in different simulation tools. Figure 4 depicts the concepts of CCSS at an abstract level. Each component is connected to an LP. The LPs are linked with each other to construct a communication backplane among co-simulation components. Each LP runs in a separate thread controlled by a local scheduler. CCSS framework implements optimal scheduling and communications scheme by using an extended version of conservative synchronization for DDES. In CCSS some LPs lag behind other LPs in time, ensuring that in spite of the full parallelization any event generated by an LP will be received at the exact time it was destined for. This results in high fidelity without compromising the simulation's speed. In addition, unlike conventional conservative approach there is no overhead of any deadlock-avoidance or deadlock-detection-and-recovery algorithm. CCSS framework ensures a synchronization that is deadlock free with no possibility of a deadlock even if some of the time-lags in the simulation are 0.

4.1.1 Calculation of Initial-Shift

As it is clear from earlier discussion, the use of initial-shift is a crucial part of CCSS framework. CCSS framework solves a system of equations to calculate initial-shifts for all of the LPs in the simulation. Lets assume L_{ij} represents the time-lag from LP i to LP j, and S_i represents the initial-shift of LP i. The initial-shifts are computed by solving this system of equations:

$$\left\{ S_i - S_j = \frac{L_{ji} - L_{ij}}{2} \;\middle|\; i < j \right\} \quad \text{where,} \quad \min_i S_i = 0. \tag{1}$$

In general, the system is converted to the form $As = b$, where $s = [S_1, S_2, \ldots, S_n]^T$, $b = [b_1, b_2, \ldots, b_m]^T$, and $A : m \times n (m \geq n)$, when initially assigning zero to S_0. Depending on the connections between the components, actually in most of the cases, this system of equations can become overdetermined. In such cases we need to remove some equations with lower preference in order to make the rank of $rank(A) = n$. A preference is a value assigned to each connection. The time-lags for the equations that were excluded are recalculated to ensure that they never exceed the minimum time-lag originally defined for those connections. The time-lags are recomputed by:

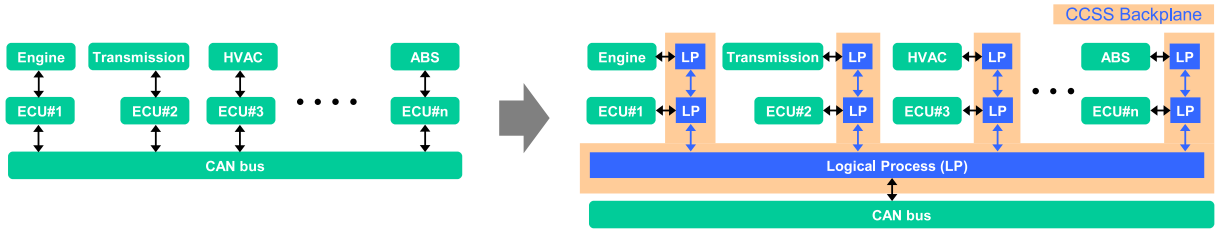

Figure 4: CCSS communication backplane.

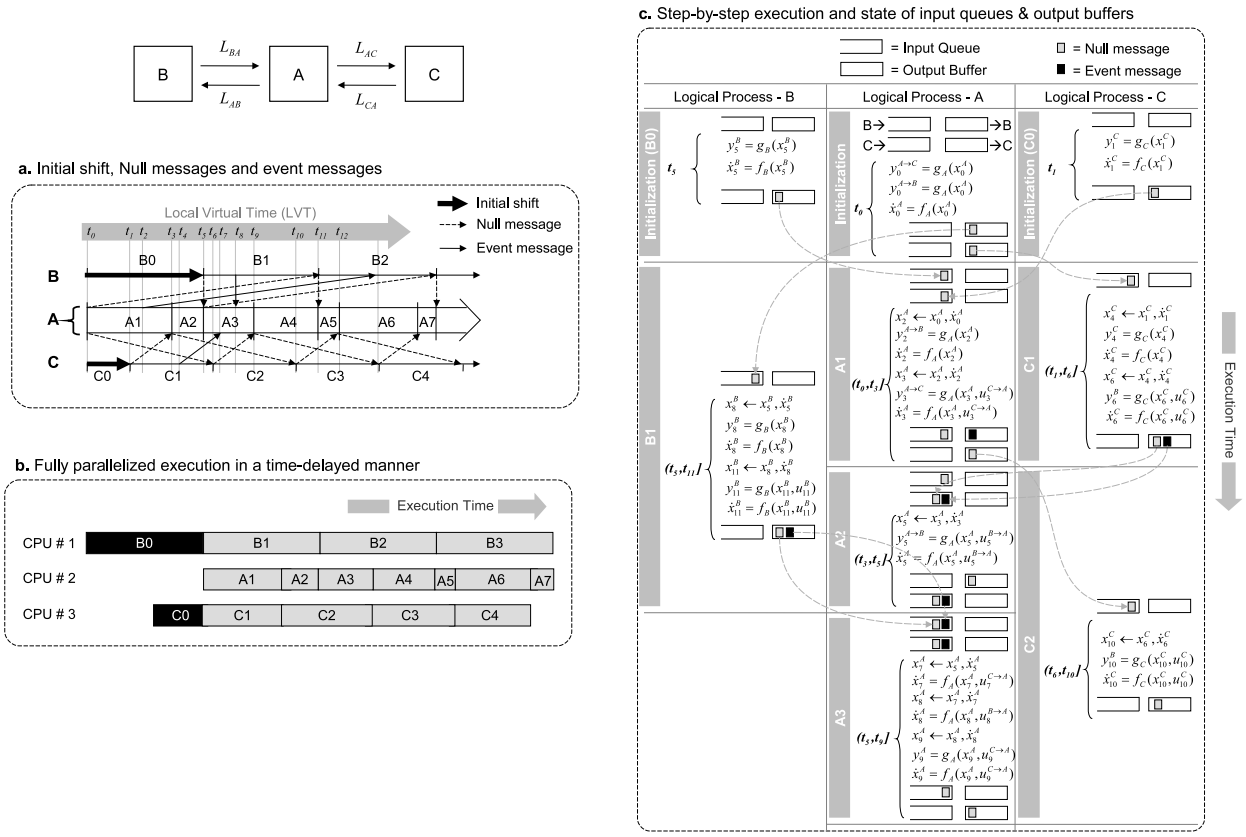

Figure 5: Different perspectives of CCSS simulation involving three logical processes: (a) Initial-shift and propagation of null and event messages (state events). Note that time-stamp of event messages are not known in advance and are only shown here for clarification, (b) Parallel execution over multiple CPUs in a time delayed manner, (c) Step-by-step execution of the simulation in mathematical form.

$$\begin{aligned} L_{ij} &:= L_{ji} + 2(S_j - S_i), \qquad \text{if } L_{ij} > L_{ji} + 2(S_j - S_i); \\ L_{ji} &:= L_{ij} + 2(S_i - S_j), \qquad \textit{otherwise}. \end{aligned} \tag{2}$$

When $(L_{ij} + L_{ji})/2 \leq 0$, then we need to merge LP i and LP j into a single simulation component. However, the discussion is skipped here because it is out of the scope of this paper.

4.1.2 Distributed Synchronization Protocol

In CCSS each LP is controlled by a local scheduler and there is no need for a centralized synchronization. Each local scheduler is responsible for managing its LVT and handling the communication with the connected LPs. The local scheduler starts by sending a null-message to all of the connected LPs after adjusting for the initial-shift. After that each LP responds to every null-message received. Null-messages are not broadcasted, instead a null-message is sent only to the source of received null-message. Event messages are generated and consumed at precise times. Using this protocol CCSS can support fully parallel execution without any compromise of simulation fidelity.

This synchronization protocol is depicted in Figure 5. The figure shows a case involving three LPs running over CCSS framework. The concept of the initial-shift is shown in Figure 5(a). The fully parallel execution in a time delayed manner can be seen in Figure 5(b). Continuous or discrete time-step functions

expect one input and generate exactly one output for each simulation step. In contrast, discrete event step functions have no such limitations. They can proceed in time without receiving any inputs. Similarly, there is no limit on the number of output events that can be generated over a given duration of time. Each event is represented by a value and an accurate timestamp. In CCSS the time always moves in a forward direction (i.e. a step is never rejected). CCSS-aware simulation modules have an ability to stop at any previously unknown time to generate an output and then resume their simulation. Figure 5(c) presents this kind of step function in a mathematical form. The state of the input and output ports is also shown before and after each time step.

From the theory of conservative synchronization in discrete event simulations we know that the simulation of an LP cannot proceed in time until all the input ports have a null or event message with a time-stamp greater than the LVT. It is clear from Figure 5(c) that there is a null-message on all of the input ports of all of the LPs at the start of each time step. For LP A, even though $L_{BA} = 0$, a null-message is always available on both of the input ports at the start of each time-step. This leads to a deadlock free 100% parallelization without any loss of fidelity.

4.2 Discussion

Now we will prove that CCSS can run high fidelity simulations without ending up in a deadlock. As is clear from Figure 5, the CCSS framework manages the communication separately for each pair of LPs. Therefore, if we can prove the condition for an arbitrary pair of LPs it will apply to the entire simulation. For the proof we consider two logical processes A and B. Both LPs have one in-port and one out-port connected with each other. The time-lag between the LPs can be defined as either "$L_{AB} = 0$, $L_{BA} > 0$" or "$0 < L_{AB} \leq L_{BA}$". The case where both time-lags are 0 results in an algebraic loop and cannot be handled by any simulation tool under normal circumstances.

Theorem 1 The simulation is 100% parallel. There are no sequential parts in the simulation and the resources are used at their full capacity.

In the theory of conservative synchronization a simulation is fully parallelized if the lookahead time is the same in both directions between two LPs. In the case of CCSS the lookahead time in both directions is always equal to $(L_{AB} + L_{BA})/2$, hence fully parallelizing the execution of the LPs on separate processing cores.

Theorem 2 The system is deadlock-free.

In a conventional conservative approach deadlock can occur when lookahead time is not equal in both directions between two connected LPs. In this situation an LP needs to wait for other LPs to finish a step. Deadlock is easy to occur in systems with complex connections especially the ones involving loops. As discussed in theorem 1, in CCSS two neighboring LPs always have equal lookahead in both directions. This leads to a deadlock free fully parallel execution even when time-lag in one direction is 0. Hence, there is no need for a deadlock avoidance or detection algorithm.

Theorem 3 Minimum number of null-messages are used for a fully parallel high fidelity implementation.

From the previous theorem we can see that the size of each time-step $T_{StepSize}$ is given by $T_{StepSize} = (L_{AB} + L_{BA})/2$ or the average of both time-lags and exactly one null-message is generated by each LP in each time-step. Now let's consider the two other possibilities and see what happens in each case.
If ($T_{StepSize} < (L_{AB} + L_{BA})/2$): The number of null-messages exchanged in a given period of time will increase as compared to CCSS. Therefore, we don't need to discuss this case.
If ($T_{StepSize} > (L_{AB} + L_{BA})/2$): Lets assume $L_{AB} > L_{BA}$. For a fully parallel implementation the step size for both the LPs must be equal. Let's assume the new step size is $Tnew_{StepSize} > (L_{AB} + L_{BA})/2$. The number of null-messages in the system will decrease but the simulation will no longer be able to support high fidelity. If LP A generates an event in the n^{th} time-step ($(t_{n-1}, t_n]$) where $t_n = n \cdot Tnew_{StepSize}$. The exact timing of the event is $t_{n-1} + \alpha$, where α is a very small positive real number. The event must be received by LP B at $t_{n-1} + \alpha + L_{AB}$. Since LP A is lagging behind LP B, the B will be executing the $(n+1)^{th}$ time-step. If

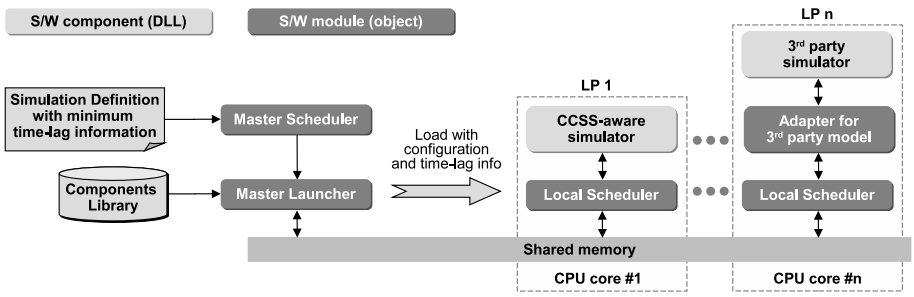

Figure 6: Implementation of CCSS framework.

$T_{StepSize} > (L_{AB} + L_{BA})/2$, then LP B will not have finished the $(n+1)^{th}$ step at time $t_{n-1} + \alpha + L_{AB}$. The LP B will not be able to consume the event at its intended time, hence losing some fidelity. This problem can be solved by running some parts of A and B sequentially.
This discussion let us conclude that $T_{StepSize} = (L_{AB} + L_{BA})/2$ is the best option since it uses the minimum number of null-messages required for a high precision fully parallel execution.

Theorem 4 The simulation is always correct and precise. The time delayed execution of the LPs does not affect the fidelity of the solution.

As discussed in theorem 3 lookahead time of $(L_{AB} + L_{BA})/2$ ensures that the events are always generated at the correct timing by the sender LP and are always received and processed at the intended point in time by the receiver LP. This ensures the simulation is always correct and the fidelity is never lost.

5 IMPLEMENTATION & RESULTS

Figure 6 describes the implementation details of the CCSS framework. The input to the system is the simulators database and Simulation Definition (SD) file. The SD lists the simulator modules used by the simulation and defines the connection information between these modules. For the sake of compatibility wrapper modules (out of the scope of this paper) are provided to convert continuous, discrete time and tool-coupling simulation models into CCSS aware discrete event simulation modules. The master scheduler calculates the initial-shift for each process by using the time-lag information provided in the SD. The master launcher then launches each LP as a thread on a processing core. Each LP is controlled by a local scheduler. The current implementation of CCSS is targeted for a multi-core processor environment and shared memory is used to communicate events among different LPs.

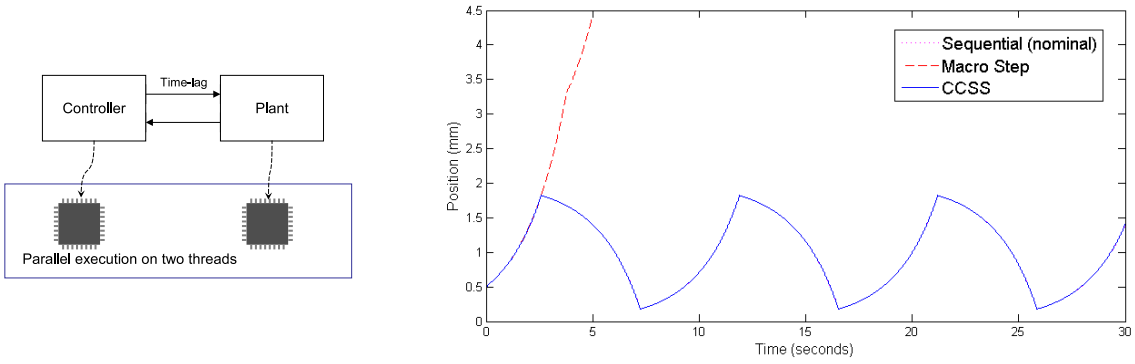

Figure 7: Delayed feedback control model.

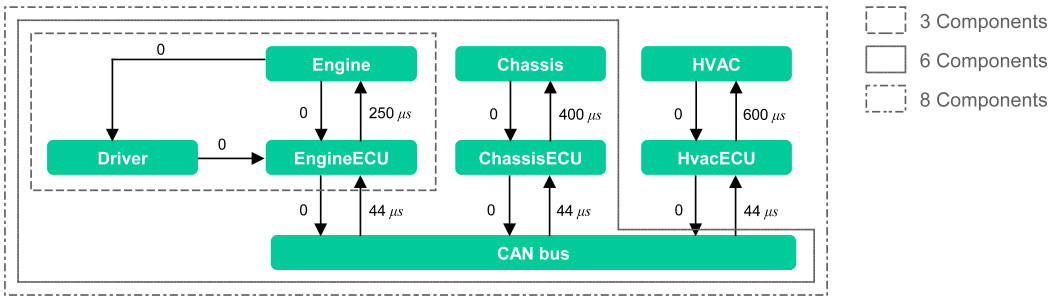

Figure 8: Integrated simulation model of a vehicle with time-lags.

For empirical results we have used two different experimental models. First model emphasizes on the importance of fidelity in a control system while the second example shows the scalability of the proposed technique by using a realistic integrated simulation scenario that can occur in an automotive simulation.

While precision can lead to error accumulation in some cases, in other cases the macro-step or any other approach that compromises on fidelity will produce entirely incorrect results. Such a scenario is shown in Figure 7. This is the output of a delayed feedback controller connected to a plant. The time-lag from the controller to the plant is $1.2ms$ and the time-lag from the plant to the controller is $0s$. The results clearly show that any approach that compromises on fidelity (e.g. the macro-step approach in the figure) fails to give the correct output. In contrast, CCSS while running in a fully parallelize manner produces the exact output without loosing any fidelity.

Empirical results for scalability were obtained by using the model shown in Figure 8. The complex control system in the figure consists of several unit control systems connected with each other via CAN. For scalability we start with only 3 components of the model, latter we show the performance of the proposed technique with 6 and 8 components. The 3 components model in the figure is an automobile model consisting of an ECU, a driver, and a plant (engine). The Driver controls the accelerator pedal in order to achieve the preset desired velocity. The ECU is responsible for calculating the throttle for the engine. The ECU also generates the fuel injection and ignition pulses. The engine component includes the powertrain and computes the current velocity, rpm, and crank angle pulse. It is important to remember that the time-lags given in the figure are the natural delays that exist between these components. Note that there is only one non-zero time-lag ($L_{EngineECU \rightarrow Engine}$). Conventional conservative synchronization will fail to achieve any parallelization. CCSS on the other hand generates a fully parallelized deadlock free synchronization with precise event handling.

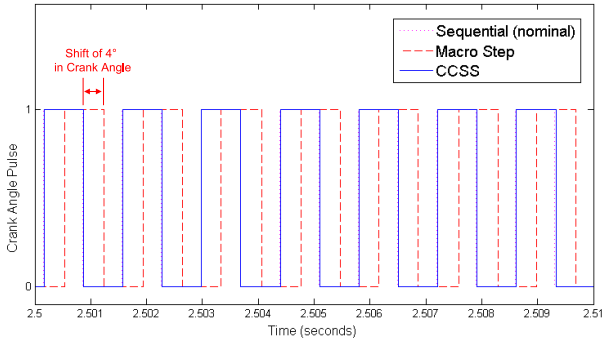

Figure 9: Crank angle error accumulation in macro-step co-simulation after just 2.5 seconds into the simulation. In contrast, CCSS exactly tracks the expected output.

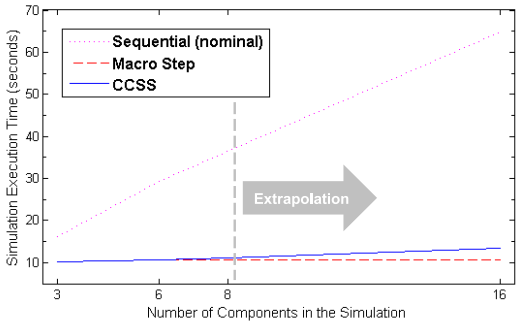

Figure 10: Simulation execution time of a 5 seconds simulation for different number of components using CCSS, Macro-Step & Sequential approach.

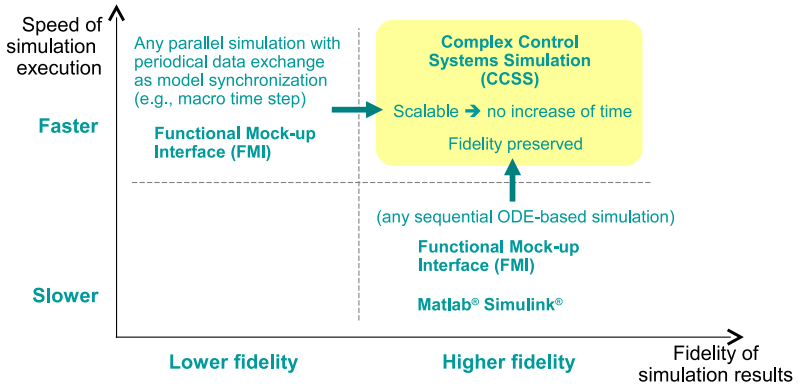

Figure 11: Advantages of CCSS. Conventional techniques either achieve high fidelity or faster execution speeds. CCSS performs high fidelity parallel simulation with no compromise on speed or fidelity.

Crank angle pulse is a very important signal that helps the ECU to compute the exact fuel injection and ignition timing for the engine. A crank angle event is generated for every 15° turn in the engine crank. In one complete revolution the pulse is generated 24 times. In order to empirically prove the high fidelity of CCSS we compare the output with macro step based co-simulation (the most often used technique for parallel execution). The nominal signal was calculated using Simulink®– *http://www.mathworks.com/products/simulink/*. From Figure 9, it is clear that the error accumulation is significant in case of macro-step co-simulation. In just 2.5 seconds of simulation there is a shift of 4° between the crank angle calculated by the macro-step approach and the nominal results (in 5 seconds this error in crank angle reaches 20°). In contrast, CCSS exactly tracks the expected output. Simulation execution speed for different number of components on Intel Xeon X5460 3.16GHz 8 cores CPU with 3GB RAM are given in Figure 10.

The scalability properties of CCSS can be deduced from Figure 9 and 10. Sequential execution produces correct output but the execution time of the simulation increases with the increase in the number of components. In case of macro-step approach the simulation execution time remains almost constant but error accumulates over time hence compromising the solution fidelity. CCSS on the other hand produces accurate results with simulation execution speed comparable to that of macro-step approach. In case of CCSS given enough processing cores the simulation execution time is only determined by the slowest component in the simulation (which can be reduced further by using *model compiler* – another tool developed by the authors – out of the scope of this paper). Advantage of CCSS over any sequential ODE-based simulation or commonly used parallel co-simulation techniques is shown in Figure 11.

6 RELATED WORK

The most commonly used technique employed in the automotive industry for parallel co-simulation is known as the fixed macro-time steps approach (Figure 12). Macro-step is an approach that simulate the communication in a simulator by fixing the synchronization points. Although this approach supports high-speed simulations, simulation errors gradually accumulate due to the large macro-step time resolution.

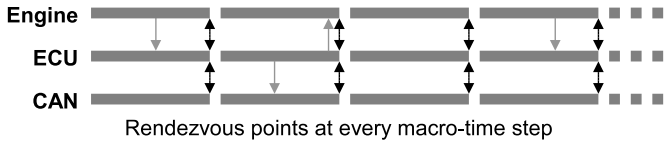

Figure 12: Macro-time step approach to co-simulation.

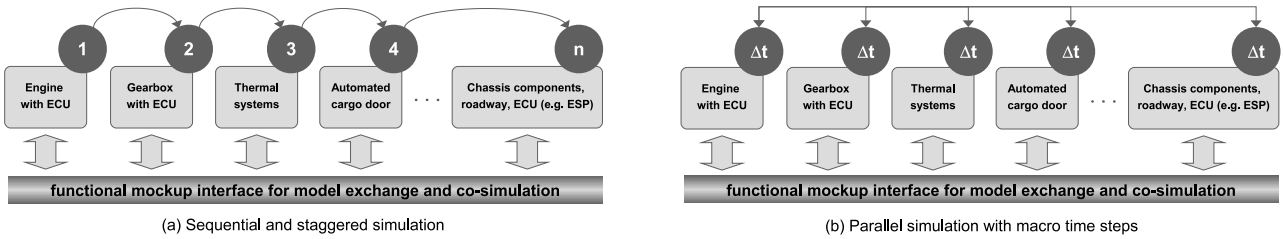

Figure 13: Co-simulation using functional mock-up interface (FMI)

Smaller macro-time steps can be used to increase the fidelity at the expense of simulation speed. Therefore, this approach compromises either speed or fidelity in the simulation.

Functional mock-up interface (FMI) – *http://functional-mockup-interface.org* is emerging as a de facto standard for the interface of simulation units in the automotive industry. FMI defines an open interface to develop complex cyber-physical systems consisting of functional mock-up units (FMU). The FMI functions are called by a simulator to create one or more instances of an FMU, called a model, and to run these models, typically together with other models. An FMU may either be self-integrating (co-simulation) or require the simulator to perform numerical integration. Alternatively, tools can be coupled via co-simulation. Even though FMI is used by a large number of automotive companies, it has several drawbacks in comparison with the approach proposed in this paper. Here are some of the trade-offs involved in the FMI:

- In *sequential and staggered simulation* (Figure 13(a)) the unit simulators are executed one by one by exchanging data in a staggered manner. In this case the elapsed time of the integrated simulation increases with the addition of new simulation units. Also, the error accumulates due to the loose coupling between the components.
- In *parallel simulation with macro time steps* (Figure 13(b)) the unit simulators are executed in parallel by exchanging the data at each macro time step. In this case the simulation fidelity may become degraded for newly added simulation units. Also there is a need to write schedulers that are specialized for the newly added units.

Mathworks Matlab® Simulink® is the industry standard for modeling control systems and is used in many engineering disciplines for modeling and simulation. Simulink essentially performs continuous or discrete-time simulation as opposed to the discrete event simulation used by the proposed CCSS framework. All of the components in the simulation follow the same time steps. For a state event (a zero-crossing event) all of the components in the simulation reject the current time step and retry the same step with a finer accuracy to achieve high fidelity. This rejection and retry policy can generate significant overhead in complex parallel simulations. Also high fidelity cannot be provided for real-time HIL simulations.

7 CONCLUSIONS & FUTURE WORK

A distributed discrete event based co-simulation technique called CCSS was proposed in this paper. CCSS extends the conventional conservative approach to distributed discrete event simulation and exploits the natural time-lag that exists in all complex control systems to enable on-time synchronization. In on-time synchronization a simulation unit can report an event at the exact time it was generated. Time delayed execution of the simulation units ensures that the receiver side simulation unit will be able to catch and process the events without waiting for synchronization points. This exploitation of the time-lag properties of the system helps CCSS achieve 100% parallel execution of different simulation units. The techniques discussed in this paper achieve high fidelity without compromising the simulation speed. CCSS is designed as a general-purpose discrete event co-simulation system with a focus on automotive industry. Although it is out of the scope of this paper, CCSS has the potential to import existing simulation units and to run them in high fidelity with all of the other benefits of CCSS.

The future direction for this work includes real-time simulation for hardware-in-loop (HIL) simulations. Accurate high-resolution real-time simulations of complex control systems will require new innovations to be incorporated with the techniques proposed in this paper.

REFERENCES

Alois, F., and T. Satish. 1994, Aug.. "Parallel and Distributed Simulation of Discrete Event Systems". Technical Report TCS-TR-3336, University of Maryland.

Bryant, R. 1984, February. "A Switch-Level Model and Simulator for MOS Digital Systems". *IEEE Transactions on Computers* C-33 (2): 160 –177.

Chandy, K., and J. Misra. 1979, Sept.. "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs". *IEEE Transactions on Software Engineering* SE-5 (5): 440 – 452.

Jefferson, D., and H. Sowizral. 1982, Dec.. "Fast Concurrent Simulation Using the Time Warp Mechanism, Part 1: Local Control". Technical Report N-1906AF, The Rand Corporation, Santa Monica, CA.

Jefferson, D., and H. Sowizral. 1985, Jan.. "Fast Concurrent Shntflafion Using the Time Warp Mechanism". In *Proceedings SCS Distributed Simulation Conference*. San Diego, CA.

Jefferson, D. R. 1985, July. "Virtual time". *ACM Transactions on Programming Languages and Systems* 7 (3): 404–425.

Misra, J. 1986. "Distributed discrete-event simulation". *ACM Computing Surveys* 18:39–65.

Perumalla, K. S. 2007. "Scaling time warp-based discrete event execution to 10^4 processors on a Blue Gene supercomputer". In *Proceedings of the 4th International Conference on Computing Frontiers*, CF '07, 69–76. New York, NY, USA: ACM.

Pretschner, A., M. Broy, I. H. Kruger, and T. Stauner. 2007. "Software Engineering for Automotive Systems: A Roadmap". In *2007 Future of Software Engineering*, FOSE '07, 55–71. Washington, DC, USA: IEEE Computer Society.

Robert Bosch GmbH 1991. *CAN Specification. Version 2.0*. Postfach 50, D-7000 Stuttgart: Robert Bosch GmbH.

Vee, V. Y., and W. J. Hsu. 1999, Aug.. "Parallel Discrete Event Simulation: A Survey". Technical report, Centre for Advanced Information Systems, Nanyang Technological University, Singapore.

AUTHOR BIOGRAPHIES

ASIM MUNAWAR is a staff researcher at IBM Research - Tokyo. He holds a M.S. and Ph.D. in Computer Science from Hokkaido University, Sapporo, Japan. His research interests include simulation technologies, stochastic optimization and GPU computing. His email address is asim@jp.ibm.com and his web page is http://www.asimmunawar.com.

TAKEO YOSHIZAWA is a staff researcher in IBM Research - Tokyo. His research interests include acceleration of simulations, parallelization algorithms and testing optimization. He holds a Master of Engineering from the University of Tokyo in Japan. His email address is ytakeo@jp.ibm.com.

TATSUYA ISHIKAWA is a researcher at IBM Research - Tokyo. He received the Master of Informatics from Kyoto University. His research interests include numerical analysis and mathematical physics. His email address is isikawa@jp.ibm.com.

SHUICHI SHIMIZU is a Senior Manager of Research at IBM Research - Tokyo. He received B.S. and M.S. degrees in Electrical Engineering from Kyoto University. His research interests lie broadly in systems engineering simulation, distributed computing, network applications, and statistical signal processing. His e-mail address is shue@jp.ibm.com.