Virtual Synchronization for Distributed Cosimulation of Dataflow Task Graphs

Seoul National Univ.
Seoul, Korea
Dohyung Kim
Contents

- Introduction
- Synchronization problem and related works
- Virtual synchronization
- Experiments
- Conclusion
Co-simulation Facts

- System level simulation is composed of multiple component simulators
- Simulation complexity is a super-linear function of design size
- Time accurate and fast simulation is the key of successful system designs
Cosimulation Speed-up

- Adopt higher abstraction level of simulation
  - Transaction level: Cadence VCC, Synopsys CoCentric, CoWare N2C
  - (cf) RTL level: MentoGrapics Seamless CVE
- Distributed simulation
  - Parallelism VS. communication overhead
- Special HW emulation board - Meerwein [6]
General Simulation Architecture

- Synchronization problem between component simulators always occurs for timed cosimulation regardless of implementations.
Synchronization Problem

Source A (8) -> HW B (4) -> SW C (1)

1 2 3 4

4th time unit

Synchronization failure!!
Related Works

- Conservative approach
  - Klein [12], Hines [13]

- Optimized approach
  - Atef [11], Sung [15]

- Optimistic approach
  - Yoo [9]
Conservative Approach

- Synchronize simulators at every simulated time
- Pros: simple
- Cons: huge synchronization overhead
Optimized Approach

- Utilize the next event time information
- Pros: reduce synchronization counts
- Cons: difficult to determine the next event time
Optimistic Approach

- Advance time first. If something goes wrong, do rollback.
- Pros: no need for the next event time prediction.
- Cons: need recovery time.

Recovery time:
- Check-pointing and state saving overhead is required.

Simulation time:
- Check-pointing and state saving overhead is required.
Proposed Approach
Virtual Synchronization

- No change of component simulator
  - We may not use an optimistic approach
- No need to predict the next event time
  - Better than the optimized approach
- BUT restriction on function execution model
Restriction on Function Execution Model

- F(X) does not depend on the arrival time of \( \{x_i\} \) but only on the arrival order of \( \{x_i\} \)
  - is a time invariant function of X

- Examples
  - Dataflow models
  - Software C-function
Execution Scenario

Optimized Approach

idle duration: local clock advancement without useful work
Virtual Synchronization Technique

- Component simulator clock is **NOT** synchronized with the global clock of the channel
- Exchanged data is still associated with the **CORRECT** time stamp

Diagram:

```
  SW simulator
     ↓
 Wrapper
     ↓
 Channel
     ↑
Wrapper
     ↑
HW simulator
```
Simulator Wrapper : Initial Iteration

1. Get the input data $x_1$ at $t_1$ time from the channel
2. Send the input data $x_1$ into the simulator
3. Acquire $\triangle (x_1)$
4. Reconstruct time accuracy by adding the arrival time $t_1$ and $\triangle (x_1)$

$\triangle (x)$ : the number of time units which simulator takes to execute data $x$
Simulator Wrapper

When $x_2$ arrived before $t_1 + \Delta(x_1)$,

$$\max(t_1 + \Delta(x_1), t_2) + \Delta(x_2)$$

When $x_2$ arrived after $t_1 + \Delta(x_1)$,

$$t_1 + \Delta(x_1) + \Delta(x_2)$$
Time Adjustment in Wrapper

Source A (8) -> HW B (4) -> SW C (1)

HW wrapper

(0,0) global time
0+4 (4) 4+1 (5,8)
8+4 (12) 12+1

Sw wrapper

local time

1 2 3 4

local time

1 2

Simulation time
Related Work: Cockx [5]

- It allows out-of-order executions of SW modules and correct execution time is recovered from later time adjustments.

- Difference
  - Applied to *delay annotated simulation* to model a SW task preemption.
  - Our approach covers system level distributed simulation including SW and HW components which use *time accurate simulators*.
Experiments
- Cosimulation Platform

- Cosimulation backplane
  - SW Simulator (Armulator)
  - HW Simulator (ModelSim)
  - Wrapper
  - Channel

TCP/IP communication
Performance Gain of Virtual Synchronization (1)

- Period: 200 ns
- Execution time: 140 ns
- Clock cycle: 20 ns

- Idle duration also includes synchronization overhead
Performance Gain of Virtual Synchronization (2)

Period: 10000 ns
Execution time: 140 ns
Clock cycle: 20 ns

Source → HW → Display

- Normal
- Optimized
- Virtual
Experiment Environment for Distributed Execution

<table>
<thead>
<tr>
<th></th>
<th>Set 1</th>
<th>Set 2</th>
<th>Set 3</th>
<th>Set 4</th>
<th>Set 5</th>
<th>Set 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph</td>
<td>(a)</td>
<td>(b)</td>
<td>(c)</td>
<td>(a), (b)</td>
<td>(a),(c)</td>
<td>all</td>
</tr>
<tr>
<td>Processor #</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>
Experiment Result for Distributed Execution

Execution time ratio: 1(SW):1(HW), 1:2, 2:1, 1:3 and 3:1

- Simulation time is bound to slowest simulator
- Scalable because of no synchronization overhead
Comparison with Seamless CVE

- **H.263 Decoder**
  - HW simulator (ModelSim) : IDCT block
  - SW simulator (Armulator) : all the other blocks

- **Seamless CVE : 2031.71 s**
  - Instruction fetch optimization
  - Data access optimization

- **Our Approach : 303s**
  - Simulation time is bound to HW simulator : 78%
Conclusion

- Virtual synchronization does not synchronize local times of each simulator directly but synchronizes times virtually by adjusting times
  - Removes synchronization overhead
  - Does not execute idle duration
  - Enable distributed execution of simulators

- Future works
  - More exact channel model
  - Multi-tasks environment for SW component