Virtual Synchronization Technique with OS Modeling for Fast and Time-accurate Cosimulation

Seoul National Univ.
Seoul, Korea
Youngmin Yi
Content

- Introduction
- Virtual Synchronization
- Motivation for OS modeling
- The proposed OS modeling
- Performance gain formula
- Experimental results
- Related works
- Conclusion
SW modules are taking more and more roles in embedded system designs.

HW/SW codesign enables concurrent design of the SW modules along with HW designs.

HW/SW cosimulation plays the key role in the HW/SW codesign.
Cosimulations in Various Stages

- **Specification**
- **Functional simulation**
- **Arch. Selection**
  - **Partitioning**
  - **Cover verification**

Cosimulation for DSE:
- Must be fast and reasonably accurate

Cosimulation for cover verification:
- Must be accurate and reasonably fast

**Boosting the time-accurate cosimulation is desirable for the last stage of DSE**
Time-accurate cosimulation consists of multiple component simulators running concurrently and interacting with each other.

We assume that a block must wait for all the input events to arrive to start its execution.
Conservative Approach

Simulators are synchronized at every simulated time.

Source A (8) → SW B (4) → HW C (1)

**synchronization point**

**n**

**n\textsuperscript{th} time unit of simulator**

**synchronization time**

Simulators are synchronized at every simulated time.
Why is Time-Accurate Cosimulation slow?

- **Low performance of the component simulator**
  - It captures all the timing information
  - Advancement of its clock actually takes time
  - ISS (instruction set simulator) for SW component
  - RTL (register transfer level) simulator for HW component

→ Virtual Synchronization

- **Time synchronization between component simulators**
  - To insure the timing correctness of the interaction between multiple component simulators

→ Virtual Synchronization
Content

- Introduction
- Virtual Synchronization
- Motivation for OS modeling
- The proposed OS modeling
- Performance gain formula
- Experimental result
- Related works
- Conclusion
Virtual Synchronization

- Component simulator clock is **NOT** synchronized with the global clock of the channel

- Exchanged data is still associated with the **CORRECT** time stamp
1. Get the input data $d_1$ at time $t_1$ from the channel
2. Send the input data $d_1$ into the simulator
3. Acquire $\Delta(d_1)$
4. Reconstruct time accuracy by adding the arrival time $t_1$ and $\Delta(d_1)$

$\Delta(x)$: the number of time units that a simulator takes to execute data $x$
Time Adjustment in Wrapper

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

SW wrapper

HW

local time

0+4 (4)

4+1

8+4 (12)

local time

simulation time
Time Adjustment in Wrapper

<table>
<thead>
<tr>
<th>Simulation Time</th>
<th>Global Time</th>
<th>Local Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW 1 2 3 4 5 6 7 8 9</td>
<td>1 2 3 4 5 6</td>
<td>1, 4+4 (4), 8+1 (8)</td>
</tr>
<tr>
<td>HW 1 2 3 4 5 6 7 8 9</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Constraint on Virtual Synch

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

\[ X=\{x_1, x_2, \ldots \} \quad Y=\{y_1, y_2, \ldots \} \]

F(x) is a time invariant function of x
Content

- Introduction
- Virtual Synchronization
- Motivation for OS modeling
- The proposed OS modeling
- Performance gain formula
- Experimental result
- Related works
- Conclusion
Motivation for OS Modeling

- **OS itself does not necessarily follow the assumption**
  - The output of a system under the control of OS will be determined by OS scheduler
  - OS does depend on the arrival time of the input data

(a) When event arrives at 10

(b) When event arrives at 15
We need OS scheduler modeling to enable virtual synchronization technique.
Motivation for OS Modeling

- If we run OS itself directly on the ISS
  ① We have to perform time synchronization in Conservative approach
  ② We have to run the OS codes
    - Context switch codes
    - Scheduling codes
    - Tick interrupt handler codes
  ③ When CPU is in the idle state, it executes idle task
  ④ To run OS on the ISS, timer interrupt must be modeled
    - Degradation of the overall simulator performance

It is very desirable to avoid running OS itself on the ISS to reduce simulation time
Content

- Introduction
- Virtual Synchronization
- Motivation for OS modeling
- The proposed OS modeling
- Performance gain formula
- Experimental result
- Related works
- Conclusion
The Proposed OS Modeling

- We execute each software task ignoring preemption possibility using virtual synchronization.

- When SW simulator returns to the wrapper, OS Modeler adjusts the time of the produced output data according to the OS scheduler policy:
  - Modeling of preemption behavior
  - Modeling of OS overheads
    - Context switch overhead
    - Tick interrupt handling overhead
1. Emit output with 
   \( t_1 + f(\Delta(d1)) \)
2. Consider the 
   preempted task if any

1. Emit output with 
   \( t_1 + f(\Delta(d1)) \)
2. \( t_2 = t_1 + f(\Delta(d1)) \)

1. Do not emit output
2. Mark \( T(d1) \) as preempted at \( t_2 \)

\[ f(t) = t + \text{RTOS overheads during } t \]
1. Emit output with \( t_1 + f(\Delta(d_1)) \)
2. Consider the preempted task if any

1. Emit output with \( t_1 + f(\Delta(d_1)) \)
2. \( t_2 = t_1 + f(\Delta(d_1)) \)

\[ \pi(task1) < \pi(task2) \]

\[ 0 < 10 < 0 + 14 \]

1. Do not emit output
2. Mark \textbf{task1} as preempted at 10
1. Emit output with $t_1 + f(\Delta(d_1))$
2. Consider the preempted task if any

1. Emit output with $t_1 + f(\Delta(d_1))$
2. $t_2 = t_1 + f(\Delta(d_1))$

- $0 < 10 < 0 + 14$
- $\pi(task1) < \pi(task2)$

1. Do not emit output
2. Mark task1 as preempted at 10
1. Emit output with \(10 + 10 = 20\)
2. Consider the preempted task if any
   1. Emit output with \(t_1 + f(\Delta(d_1))\)
   2. \(t_2 = t_1 + f(\Delta(d_1))\)
1. Emit output with \(10+10 = 20\)
2. Consider the preempted task1

1. Emit output with \(t1+f(\Delta(d1))\)
2. \(t2 = t1+f(\Delta(d1))\)

\[10 < t2 < 10 + (7+3)\]

\[\pi(\tau(d1)) < \pi(\tau(d2))\]

1. Do not emit output
2. Mark \(T(d1)\) as preempted at \(t2\)
The RTOS Overheads

- **RTOS overheads under consideration**
  - Context switch and scheduler codes
    - when a task voluntarily yields the control
    - when a task is preempted by another task
  - Timer interrupt handler

- The execution time of the timer interrupt handler varies slightly depending on the number of sleeping tasks

- The estimated values are obtained from the library after the initial measurement
The Accuracy of the OS Modeling

- **Accuracy of this approach is dependent on the accuracy of the estimated RTOS overheads**
  - RTOS has the constant or bounded overhead
- **Cache is a major source of inaccuracy**
  - Cache related preemption delay
  - Smaller cache misses for the second instance when it preempts some task consecutively
  - This approach cannot accurately model the effect of temporal locality

![Diagram showing the contrast between reality and the proposed approach before time adjustment](image-url)
Content

- Introduction
- Virtual Synchronization
- Motivation for OS modeling
- The proposed OS modeling
- Performance gain formula
- Experimental result
- Related works
- Conclusion
Performance Gain Formula

- The expected performance gain =
  \[
  1 - \frac{ST_{model}}{ST_{RTOS}}
  \]

- Notations
  - \( r \) : the ratio of the RTOS overheads to the total simulation time
  - \( i \) : the ratio of the CPU idle duration to the total simulation time
  - \( s \) : the degradation ratio of simulation time to advance one simulated cycle

\[
1 - \frac{ST_{model}}{ST_{RTOS}} = \frac{s + r + i}{1 + s}
\] (eq.1)
Content

- Introduction
- Virtual Synchronization
- Motivation for OS modeling
- The proposed OS modeling
- Performance gain formula
- Experimental result
- Related works
- Conclusion
Experiments: Configuration

- **Video phone application (4 tasks)**
  - H.263 encoder, H.263 decoder
  - G.723 encoder, G.723 decoder

- **Two RTOSes**
  - eCOS, uCOS-ii

- **ARMulator**
  - ARM720T with 8KB unified cache
Experiments: Performance Results

due to the removal of the time synchronization

due to the OS modeling itself

- Time consumed to simulate uCOS-ii
- Processor utilization is 73.0%
Experiments: Performance Results

- Performance improvement due to OS modeling itself

<table>
<thead>
<tr>
<th></th>
<th>Estimated</th>
<th>Measured</th>
<th>(eq 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$s$</td>
<td>$i$</td>
<td>$r$</td>
</tr>
<tr>
<td>eCOS</td>
<td>15 %</td>
<td>4.2%</td>
<td>3.4%</td>
</tr>
<tr>
<td>uCOS</td>
<td>15 %</td>
<td>27.0%</td>
<td>1.0%</td>
</tr>
</tbody>
</table>

$$1 - \frac{ST_{model}}{ST_{RTOS}} = \frac{s + r + i}{1 + s}$$  \hspace{1cm} (eq.1)
Experiments: Accuracy Results

- **Metric for the accuracy**

\[
error = \frac{\text{execution}_{\text{model}} - \text{execution}_{\text{RTOS}}}{\text{execution}_{\text{RTOS}}}
\]

<table>
<thead>
<tr>
<th>Task</th>
<th>Instances</th>
<th>Cache Disabled</th>
<th>Cache Enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>eCOS</td>
<td>uCOS</td>
</tr>
<tr>
<td>G.723 dec</td>
<td>100</td>
<td>0.06 %</td>
<td>-0.02 %</td>
</tr>
<tr>
<td>G.723 enc</td>
<td>100</td>
<td>-0.02 %</td>
<td>-0.05 %</td>
</tr>
<tr>
<td>H.263 dec</td>
<td>10</td>
<td>-0.03 %</td>
<td>-0.03 %</td>
</tr>
<tr>
<td>H.263 enc</td>
<td>10</td>
<td>0.08 %</td>
<td>0.01 %</td>
</tr>
</tbody>
</table>
Related Work

- **RTOS modeling for transaction level cosimulation with delay annotation**
  - Pia[5]
  - POLIS[3]
  - Cockx[4]
  - Yoo et al[6]
Conclusion

- We extend the virtual synchronization technique with OS modeling
- OS modeler models the RTOS overheads as well as preemption behavior
- With combined benefits from OS modeling and virtual synchronization, we could reduce simulation time while preserving the accuracy
  - By removing time synchronization overhead
  - By avoiding the execution of idle task
  - By not modeling timer interrupt inside the simulator
Thank you