

Out-of-Order Parallel Simulation of SystemC Models using the RISC Framework

*Tutorial at Embedded Systems Week 2020
September 20, 2020*

Rainer Dömer
doemer@uci.edu

Center for Embedded and Cyber-Physical Systems
University of California, Irvine

UCIrvine
University of California, Irvine



Virtual Tutorial Logistics

- ESWeek 2020 is a Virtual Conference
 - This tutorial is virtual, too!
- Zoom Meeting ID: 985 6461 2963
 - 9:00 AM - 1:00 PM EDT (GMT-4)
 - <https://uci.zoom.us/j/98564612963>
 - No meeting passcode needed
- Live Presentation via Zoom
 - Session will be recorded for offline reference
 - Presenter: Live on camera
 - Audience: *Please turn on your camera as well*
 - Request-but-not-require policy
- Interactive Discussion: *Please Participate*
 - Chat, Raise hand, Polls, Breakout rooms, Reactions...

Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

2

Poll: Time Zone

- Which time zone are you in?
(Choose the closest one)
[Single Choice]
 - Answer 1: PDT (Los Angeles) UTC-7
 - Answer 2: EDT (New York) UC-4
 - Answer 3: CEST (Paris) UTC+2
 - Answer 4: IST (New Delhi) UTC+5:30
 - Answer 5: CST (Beijing) UTC+8
 - Answer 6: AEST (Sydney) UTC+10

Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

3

Agenda

- Part 1: 9:00am EDT
Introduction to Out-of-Order Parallel Discrete Event Simulation
- Part 2: 9:40am EDT
Overcoming the Obstacles of IEEE SystemC Semantics
- Part 3: 10:20am EDT
RISC: Recoding Infrastructure for SystemC
- Part 4: 11:15am EDT
**Hands-on Practical Training
with RISC Compiler and Simulator**
- Part 5: 12:15pm EDT
**Hands-on Practical Analysis
of Parallel Potential of SystemC Models**
 - Note: For hands-on participation, you will need a Linux account on a multi-core host with Docker access to download RISC (otherwise you are welcome to just watch the demos)

Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

4

Out-of-Order Parallel Simulation of SystemC Models using the RISC Framework

Part 1: Introduction to Out-of-Order Parallel Discrete Event Simulation

Rainer Dömer
doemer@uci.edu

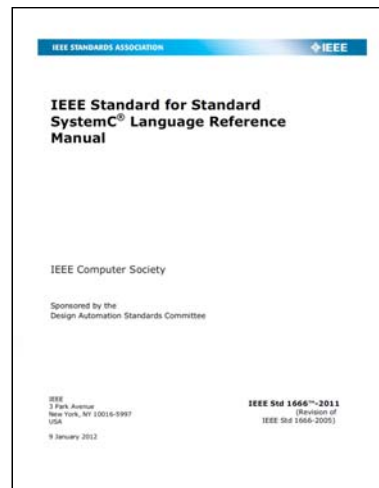
Center for Embedded and Cyber-Physical Systems
University of California, Irvine

UCIrvine
University of California, Irvine



IEEE Standard 1666-2011

- The SystemC Language
 - Official standard
 - De-facto standard
- ... for
 - Modeling and
 - Simulation
- ... of systems containing
 - Hardware and
 - Software
- Discrete Event Simulation
 - Accellera (sequential)
 - RISC (parallel)



Discrete Event Simulation (DES)

- SystemC uses Traditional DES
 - Concurrent threads of execution
 - Managed by a central scheduler
 - Driven by events and time advances
 - Delta cycle
 - Time cycle
 - Partial temporal order with barriers
- Cooperative Multi-Tasking
 - IEEE 1666-2011 standard
 - A single thread is active at any time
 - Does not exploit parallelism
 - Cannot utilize multiple cores
 - Sequential simulation is slow

Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

7

Discrete Event Simulation (DES)

- Specific Example:
Accellera SystemC
Proof-of-Concept Library
- uses an extra **root thread** for the following tasks:
 - Elaboration phase
 - Scheduling
 - Event notifications
 - Channel updates
 - Delta cycle updates
 - Simulation time updates
 - SC_METHOD calls
 - (not shown)

Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

8

Approaches for Faster Simulation

Improved Modeling Techniques

- Transaction-level modeling (TLM)
- TLM temporal decoupling
- Savoiu et al. [MEMOCODE'05]
- Razaghi et al. [ASPDAC'12]

Distributed Simulation

- Chandy et al. [TSE'79]
- Huang et al. [SIES'08]
- Chen et al. [CECS'11]

Sequential DE simulation is slow

Hardware-based Acceleration

- Sirowy et al. [DAC'10]
- Nanjundappa et al. [ASPDAC'10]
- Sinha et al. [ASPDAC'12]

SMP Parallel Simulation

- Fujimoto [CACM'90]
- Chopard et al. [ICCS'06]
- Ezudheen et al. [PADS'09]
- Mello et al. [DATE'10]
- Schumacher et al. [CODES'11]
- Chen et al. [TCAD'14]
- Yun et al. [TCAD'12]
- Schmidt et al. [DAC'17]
- and many others

Tutorial at ESWEEK, Sept. 20, 2020
(c) 2020 R. Doemer et al., CECS
9

Discrete Event Simulation (DES)

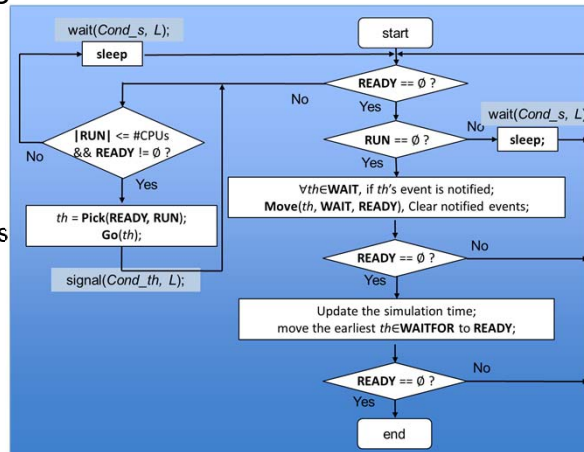
- Traditional DES Algorithm (sequential)
 - Active threads are managed in a READY queue
 - Waiting threads are managed in WAIT queues
 - wait(event);
 - wait(time);
 - Simulation progress
 - Delta cycle
 - Time cycle
 - Scheduler picks a *single* thread and executes it

Tutorial at ESWEEK, Sept. 20, 2020
(c) 2020 R. Doemer et al., CECS
10

Parallel Discrete Event Simulation (PDES)

Parallel DES Algorithm

- Active threads are managed in a READY queue
- Waiting threads are managed in WAIT queues
- Simulation progress
 - Delta cycle
 - Time cycle
- Scheduler picks N threads and executes them *in parallel*
- N = number of available CPU cores



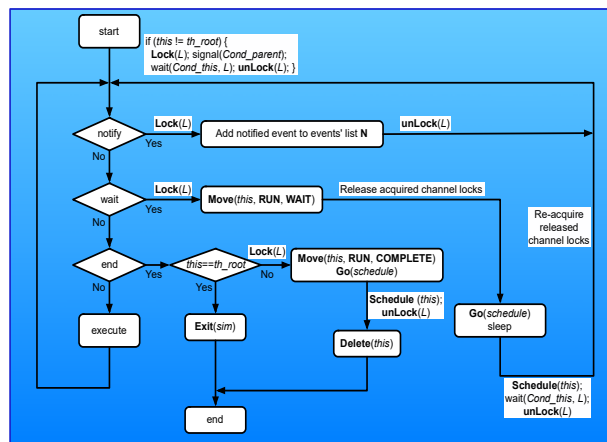
Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

11

Parallel Discrete Event Simulation (PDES)

- Parallel DES Algorithm requires safe synchronization
 - Locks and condition variables (e.g. POSIX multi-threading)
- Protected scheduling resources
- Protected communication
- MT-safe SystemC primitives
- Example: Life-cycle of a SC_THREAD



Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

12

Parallel Discrete Event Simulation (PDES)

- **Parallel DES [Fujimoto1990]**
 - Threads execute in parallel *iff*
 - in the same delta cycle, *and*
 - in the same time cycle
 - Significant speed up!
 - Cycle boundaries are absolute barriers: *Synchronous* PDES
- **Aggressive *Asynchronous* PDES**
 - Conservative Approaches
 - Careful static analysis prevents conflicts
 - Optimistic Approaches
 - Conflicts are detected and addressed (*roll back*)

Tutorial at ESWEEK, Sept. 20, 2020 (c) 2020 R. Doemer et al., CECS 13

Parallel Discrete Event Simulation (PDES)

- **Out-of-Order Parallel DES**
 - Threads execute in parallel *iff*
 - in the same delta cycle, *and*
 - In the same time cycle,
 - **OR if there are no conflicts!**
 - Breaks synchronization barrier
 - Threads run as soon as possible, even ahead of time
 - Results in even higher speedup!
 - [DATE'12], [IEEE TCAD'14]
 - Needs compiler support for data and event **conflict analysis!**
 - Preserve the accuracy of cause and effect relationship
 - Accurate results, accurate timing

Tutorial at ESWEEK, Sept. 20, 2020 (c) 2020 R. Doemer et al., CECS 14

Summary and Analysis

- Traditional Discrete Event Simulation (DES)
 - Simulator runs *sequentially*, executes one thread at a time
 - Cannot utilize the parallelism of multi- or many-core hosts
- Parallel Discrete Event Simulation (PDES)
 - Threads run in *parallel* (if at the same delta and time cycle)
 - Simulation cycles are absolute barriers
- Out-of-order Parallel DE Simulation (OoO PDES)
 - Non-conflict threads run in *parallel and ahead-of-time* [DATE'12]
 - Maximum parallelism, order of magnitude speedup! [TCAD'14]
- Problem solved!? *Not quite!*
- What about host platforms? *Readily available.*
- What about accuracy? *Achievable with careful analysis.*
- What about standard compliance? *That's where the problem is!*

Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

15

Problem Definition

- What is given?
 - Embedded systems are parallel
 - SystemC is suitable and standard for system design
 - Models exhibit explicit thread-level parallelism
 - Multi- and many-core host platforms are readily available
- What do we want?
 - Fastest Parallel Discrete Event Simulation
 - For the SystemC language
- What is the objective?
 - Maximize compliance with the IEEE 1666-2011 standard
- Why is this so difficult?
 - There are “*Seven Obstacles in the Way of Standard-Compliant Parallel SystemC*” [ESL'16]

Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

17

Out-of-Order Parallel Simulation of SystemC Models using the RISC Framework

Part 2: Overcoming the Obstacles of IEEE SystemC Semantics

Rainer Dömer
doemer@uci.edu

Center for Embedded and Cyber-Physical Systems
University of California, Irvine



Compliance with IEEE SystemC Semantics

- IEEE Standard 1666™-2011
 - Revision of IEEE Std. 1666-2005
 - Standard SystemC® Language Reference Manual
- ... unfortunately stands in the way of parallel SystemC simulation!
- SystemC Evolution Day 2016
 - “Seven Obstacles in the Way of Parallel SystemC Simulation”, Rainer Doemer, Munich, Germany, May 2016.
 - SystemC standard
 - ... must embrace true parallelism
 - ... must evolve in a major revision (3.x)



Poll: SystemC Evolution

- Take a guess, what happened?
[Single Choice]
 - Answer 1: The speaker was thrown off the stage
 - Answer 2: SystemC Revolution:
Major changes to SystemC standard
 - Answer 3: SystemC Evolution:
Minor changes to SystemC standard
 - Answer 4: Nothing, SystemC standard didn't change.
 - Answer 5: Parallel simulation of SystemC changed.

Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS 20


Compliance with IEEE SystemC Semantics

- SystemC Evolution!?
 - Nothing substantial has changed...
- In absence of major changes to SystemC standard, my group worked hard on the compliance problem
 - *"Let's make the best of it!"*
 - Accept SystemC standard as it is (well, most of it)
 - Build the best parallel SystemC simulator possible
 - Aim for maximum compliance with the standard
 - We took this risk, and created RISC!
 - *Recoding Infrastructure for SystemC*
 - RISC pushes the limits to overcome the 7 obstacles!

Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS 21

Obstacle 1: Co-Routine Semantics



- Fact: IEEE 1666-2011 requires *co-operative multitasking*
 - Quotes from Section “4.2.1.2 Evaluation phase” (pages 17, 18):

Since process instances execute without interruption, **only a single process instance can be running at any one time**. [...] A process shall not pre-empt or interrupt the execution of another process. This is known as **co-routine semantics** or **co-operative multitasking**. [...]

The scheduler is **not pre-emptive**. An application can assume that a method process will execute in its entirety without interruption, and a thread or clocked thread process will execute the code **between two consecutive calls to function wait without interruption**.
- Problem: Uninterrupted execution guarantee

An implementation running on a machine that provides hardware support for concurrent processes may permit two or more processes to run concurrently, provided that the behavior appears identical to the co-routine semantics defined in this subclause. In other words, the implementation would be obliged to analyze any dependencies between processes and to constrain their execution to match the co-routine semantics.

Tutorial at ESWEEK, Sept. 20, 2020 (c) 2020 R. Doemer et al., CECS 22

Obstacle 1: Co-Routine Semantics

- Parallel DES [Fujimoto 1990]
 - Threads execute in parallel *iff*
 - in the same delta cycle, *and*
 - in the same time cycle
 - Order of magnitude speed up!
- IEEE 1666 Requirement:

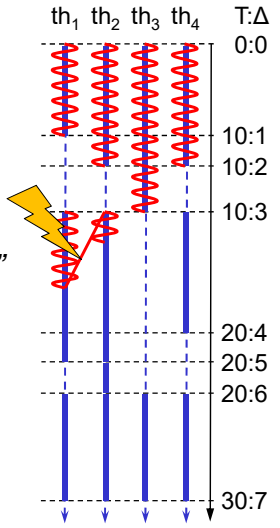
“The scheduler is not pre-emptive.”

```
int x; // shared variable

void thread1()      void thread2()
{ x = 0;            { x = 7;
  x = x + 1;        { x = x * 6;
  cout << x;        cout << x;
}                  }

```

 - SystemC: guaranteed safe!
 - PDES: not safe! (race condition)



Tutorial at ESWEEK, Sept. 20, 2020 (c) 2020 R. Doemer et al., CECS 23

Obstacle 1: Co-Routine Semantics

- Fact: IEEE 1666-2011 requires *co-operative multitasking*

➤ Quotes from Section "4.2.1.2 Evaluation phase" (pages 17, 18):

Since process instances execute without interruption, **only a single process instance can be running at any one time**, [...]. A process shall not pre-empt or interrupt the execution of another process. This is known as *co-routine semantics* or *co-operative multitasking*.
[...]

The scheduler is **not pre-emptive**. An application can assume that a method process will execute in its entirety without interruption, and a thread or clocked thread process will execute the code **between two consecutive calls to function wait without interruption**.

- Problem: Uninterrupted execution guarantee

An implementation running on a machine that provides hardware support for concurrent processes may permit two or more processes to run concurrently, provided that the behavior appears identical to the co-routine semantics defined in this subclause. In other words, the implementation would be obliged to **analyze any dependencies** between processes and to constrain their execution to **match the co-routine semantics**.

- Proposal: Explicitly allow parallel execution, preemption
 - Process instances at the same time (t, δ) may execute in parallel
 - Model designer must write thread safe code, avoid race conditions
 - Parallel systems, parallel models, parallel programming

Overcoming the Obstacles

- Obstacle 1:
Resolved!

➤ Introduce a dedicated SystemC Compiler

➤ Automatic analysis of parallel access conflicts

➤ Run SystemC processes in parallel if there are no conflicts
➤ Faster simulation
➤ Results remain the same

Obstacle 1: Co-Routine Semantics

- Fact: IEEE 1666-2011 requires *co-operative multitasking*

➤ Quotes from Section "4.2.1.2 Evaluation phase" (pages 17, 18):

Since process instances execute without interruption, **only a single process instance can be running at any one time**, [...]. A process shall not pre-empt or interrupt the execution of another process. This is known as *co-routine semantics* or *co-operative multitasking*.
[...]

The scheduler is **not pre-emptive**. An application can assume that a method process will execute in its entirety without interruption, and a thread or clocked thread process will execute the code **between two consecutive calls to function wait without interruption**.

- Problem: Uninterrupted execution guarantee

An implementation running on a machine that provides hardware support for concurrent processes may permit two or more processes to run concurrently, provided that the behavior appears identical to the co-routine semantics defined in this subclause. In other words, the implementation would be obliged to **analyze any dependencies** between processes and to constrain their execution to **match the co-routine semantics**.

- Proposal: Explicitly allow parallel execution, preemption
 - Process instances at the same time (t, δ) may execute in parallel
 - Model designer must write thread safe code, avoid race conditions
 - Parallel systems, parallel models, parallel programming

Obstacle 2: Simulator State

- Fact: Discrete Event Simulation (DES) is presumed
 - Example from IEEE 1666-2011, page 31: `sysc/kernel/sc_simcontext.h`

```
[...]  
bool sc_pending_activity_at_current_time();  
bool sc_pending_activity_at_future_time();  
bool sc_pending_activity();  
bool sc_time_to_pending_activity();  
[...]
```
- Problem: Parallel Discrete Event Simulation (PDES) is different from sequential DES
 - After elaboration, there may be *multiple running threads*
 - Scheduling may happen while some threads are still running
- Proposal: Carefully review simulator state primitives and revise as needed for PDES
 - Adapt the functions and APIs for parallel execution semantics
 - The general notion of *shared state* needs attention...

Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

26

Overcoming the Obstacles

- Obstacle 2: *Ongoing...*
 - Review and revise the SystemC API
 - Slightly adjust the semantics
 - Maximize compliance with standard
 - For APIs on the slide:
 - Users' expectations can be met
 - Example: SystemC integration with virtual platforms works fine

Obstacle 2: Simulator State

- Fact: Discrete Event Simulation (DES) is presumed
 - Example from IEEE 1666-2011, page 31: `sysc/kernel/sc_simcontext.h`

```
[...]  
bool sc_pending_activity_at_current_time();  
bool sc_pending_activity_at_future_time();  
bool sc_pending_activity();  
bool sc_time_to_pending_activity();  
[...]
```
- Problem: Parallel Discrete Event Simulation (PDES) is different from sequential DES
 - After elaboration, there may be *multiple running threads*
 - Scheduling may happen while some threads are still running
- Proposal: Carefully review simulator state primitives and revise as needed for PDES
 - Adapt the functions and APIs for parallel execution semantics
 - The general notion of *shared state* needs attention...

Seven Obstacles in the Way of Parallel SystemC Simulation

(c) 2016 R. Doemer, CECS

10

Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

27

Obstacle 3: Lack of Thread Safety

- Fact: Primitives are generally not multi-thread safe
 - Suspicious example from IEEE 1666-2011, page 194:


```
[...]
sc_length_param  length10(10);
sc_length_context cntxt10(length10); // length10 now in context
sc_int_base      int_array[2];      // Array of 10-bit integers
[...]
```
- Problem: Parallel execution may lead to race conditions
 - Race conditions result in non-deterministic/undefined behavior
 - Explicit protection (e.g. by mutex locks) is cumbersome
 - Identifying problematic constructs is difficult
 - Example: `class sc_context`, commented as “co-routine safe”
- Proposal: Require *all* primitives to be multi-thread safe
 - Carefully revise the proof-of-concept SystemC library
 - Encouraging item: `async_request_update` is thread-safe!
 - See “5.15 sc_prim_channel”, IEEE 1666-2011, page 121

Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

28

Overcoming the Obstacles

- Obstacle 3: *Ongoing...*
- Revise SystemC primitives for multi-thread safety
 - Protection by inserted locks
 - Store state in local or thread-local storage
 - For deterministic debugging, user can control number of parallel threads (e.g. set to 1)

Obstacle 3: Lack of Thread Safety

- Fact: Primitives are generally not multi-thread safe
 - Suspicious example from IEEE 1666-2011, page 194:


```
[...]
sc_length_param  length10(10);
sc_length_context cntxt10(length10); // length10 now in context
sc_int_base      int_array[2];      // Array of 10-bit integers
[...]
```
- Problem: Parallel execution may lead to race conditions
 - Race conditions result in non-deterministic/undefined behavior
 - Explicit protection (e.g. by mutex locks) is cumbersome
 - Identifying problematic constructs is difficult
 - Example: `class sc_context`, commented as “co-routine safe”
- Proposal: Require *all* primitives to be multi-thread safe
 - Carefully revise the proof-of-concept SystemC library
 - Encouraging item: `async_request_update` is thread-safe!
 - See “5.15 sc_prim_channel”, IEEE 1666-2011, page 121

Seven Obstacles in the Way of Parallel SystemC Simulation

(c) 2016 R. Doemer, CECS

11

Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

29

Obstacle 4: Class `sc_channel`

- **Fact:** `sc_channel` is an alias type for `sc_module`
 - IEEE 1666-2011, Section “5.2.23 `sc_behavior` and `sc_channel`” (page 56):

The typedefs `sc_behavior` and `sc_channel` are provided for users to express their intent.
NOTE—There is **no distinction between a *behavior* and a hierarchical channel** other than a difference of intent. Either may include both ports and public member functions.
 - `systemc-2.3.1/include/sysc/kernel/sc_module.h`

```
[...]
typedef sc_module sc_channel;
typedef sc_module sc_behavior;
[...]
```
- **Problem:** Alias type is only another name, no new type
 - Language does not distinguish modules and channels
 - No separation of communication and computation
 - Breaks a key system-level design principle...
- **Proposal:** Class `sc_channel`, derived from `sc_module`
 - Module encapsulates computation (hosts threads/processes)
 - Channel encapsulates communication (implemented interfaces)

Tutorial at ESWEEK, Sept. 20, 2020 (c) 2020 R. Doemer et al., CECS 30

Overcoming the Obstacles

- Obstacle 4:
Fixed!
- Derive `sc_channel` from base class `sc_module`
 - Minimal change in SystemC headers
 - Two different types at compile-time
 - Easy distinction in static analysis
 - No known negative side-effects

Obstacle 4: Class `sc_channel`

- **Fact:** `sc_channel` is an alias type for `sc_module`
 - IEEE 1666-2011, Section “5.2.23 `sc_behavior` and `sc_channel`” (page 56):

The typedefs `sc_behavior` and `sc_channel` are provided for users to express their intent.
NOTE—There is **no distinction between a *behavior* and a hierarchical channel** other than a difference of intent. Either may include both ports and public member functions.
 - `systemc-2.3.1/include/sysc/kernel/sc_module.h`

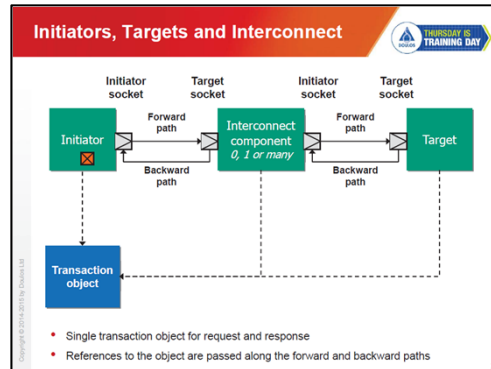
```
[...]
typedef sc_module sc_channel;
typedef sc_module sc_behavior;
[...]
```
- **Problem:** Alias type is only another name, no new type
 - Language does not distinguish modules and channels
 - No separation of communication and computation
 - Breaks a key system-level design principle...
- **Proposal:** Class `sc_channel`, derived from `sc_module`
 - Module encapsulates computation (hosts threads/processes)
 - Channel encapsulates communication (implemented interfaces)

Seven Obstacles in the Way of Parallel SystemC Simulation (c) 2016 R. Doemer, CECS 12

Tutorial at ESWEEK, Sept. 20, 2020 (c) 2020 R. Doemer et al., CECS 31

Obstacle 5: TLM-2.0

- Fact: Channel concept has disappeared
 - "The Definitive Guide to SystemC: TLM-2.0 and the IEEE 1666-2011 Standard", Presentation by David Black, Doulos, at DAC'15 Training Day
- Problem: Where is the channel?

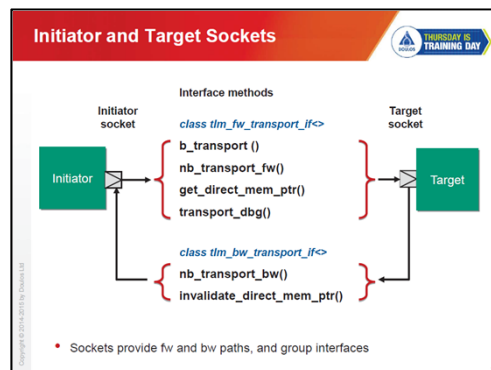


Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS 32

Obstacle 5: TLM-2.0

- Fact: Channel concept has disappeared
 - "The Definitive Guide to SystemC: TLM-2.0 and the IEEE 1666-2011 Standard", Presentation by David Black, Doulos, at DAC'15 Training Day
- Problem: Where is the channel?
 - Interface methods are well-defined, but not contained
 - Separation of concerns "Computation ≠ Communication" principle is broken
 - Proposal: Encapsulate communication methods in channels

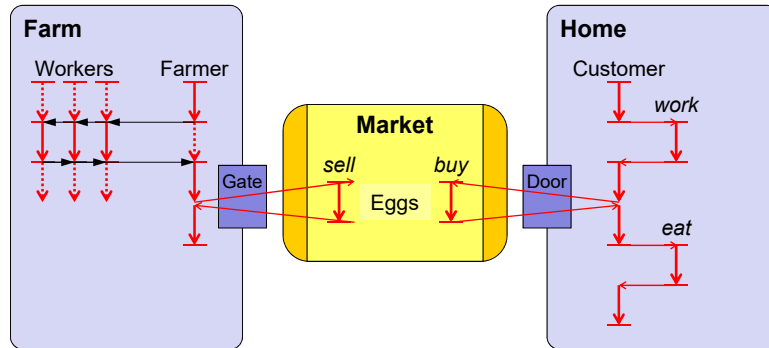


Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS 33

Overcoming Obstacle 5

- Classic TLM: Producer-Consumer Example



- Modules wrap computation, channels wrap communication
- Threads operate in their own modules or protected channels
- Well-behaved execution in safe execution contexts

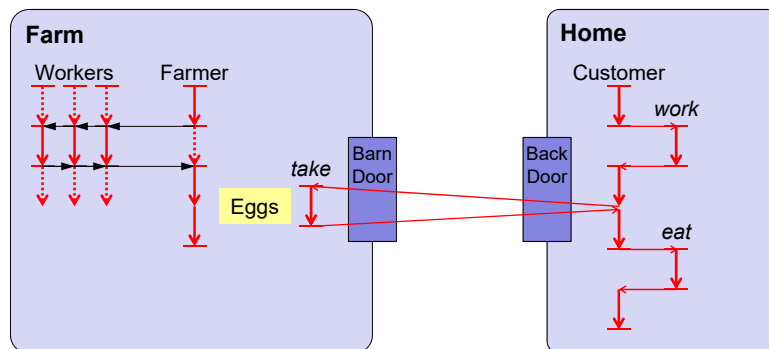
Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

34

Overcoming Obstacle 5

- New TLM-2.0: Producer-Consumer Example



- No channels! Threads operate directly in others' modules
- Fast, but dangerous execution in foreign territory
- Requires deep analysis and well-designed models

Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

35

Overcoming the Obstacles

- Obstacle 5:
*Reevaluated,
Resolved!*
- Socket Call Path (SCP) analysis
- Variable Entanglement analysis
 - Compile-time analysis can identify target methods executed by TLM-2.0 calls
 - Support for interconnect modules and DMI [CODES+ISSS'19, ACM TECS]

Obstacle 5: TLM-2.0

- Fact: Channel concept has disappeared
 - "The Definitive Guide to SystemC: TLM-2.0 and the IEEE 1666-2011 Standard", Presentation by David Black, Doulos, at DAC'15 Training Day
- Problem: Where is the channel?
 - Interface methods are well-defined, but not contained
 - Separation of concerns "Computation ≠ Communication" principle is broken
 - Proposal: Encapsulate communication methods in channels

Seven Obstacles in the Way of Parallel SystemC Simulation (c) 2016 R. Doemer, CECS 13

Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

36

Obstacle 6: Sequential Mindset

- Fact: SC_METHOD is preferred over SC_THREAD, context switches are considered overhead
 - IEEE 1666-2011, Section 5.2.11 on threads (page 44):

Each thread or clocked thread process requires its own execution stack. As a result, context switching between thread processes may impose a simulation overhead when compared with method processes.
- Problem: Sequential modeling is encouraged
 - However, systems are parallel by nature, so should be models
 - Avoiding context switches is the wrong optimization criterion
- Proposal: Use actual threads, eliminate SC_METHOD, identify dependencies among threads
 - Promote parallel mindset, with true thread-level parallelism
 - Speed due to parallel execution, not due to fewer context switches
 - Explicitly express task relations (use `e.notify()`, `wait(e)`)
 - Synchronize, communicate through events and channels

Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

37

Overcoming the Obstacles

- Obstacle 6:
Not a problem

➤ **SC_METHOD,**
SC_THREAD,
SC_CTHREAD
can all be supported

- Static analysis per process type
- **SC_METHOD** execution by dedicated invoker threads
- Nice optimization problem for efficient grouping with minimal conflicts

Obstacle 6: Sequential Mindset

- Fact: SC_METHOD is preferred over SC_THREAD, context switches are considered overhead
 - IEEE 1666-2011, Section 5.2.11 on threads (page 44):
Each thread or clocked thread process requires its own execution stack. As a result, context switching between thread processes may impose a simulation overhead when compared with method processes.
- Problem: Sequential modeling is encouraged
 - However, systems are parallel by nature, so should be models
 - Avoiding context switches is the wrong optimization criterion
- Proposal: Use actual threads, eliminate SC_METHOD, identify dependencies among threads
 - Promote parallel mindset, with true thread-level parallelism
 - Speed due to parallel execution, not due to fewer context switches
 - Explicitly express task relations (use `e.notify()`, `wait(e)`)
 - Synchronize, communicate through events and channels

Seven Obstacles in the Way of Parallel SystemC Simulation (c) 2016 R. Doemer, CECS 14

Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

38

Obstacle 7: Temporal Decoupling

- Fact: TD is designed to speed up sequential DES
 - IEEE 1666-2011, Section 12.1 on "TLM-2.0 global quantum" (page 453):
Temporal decoupling permits SystemC processes to run ahead of simulation time for an amount of time known as the time quantum and is associated with the loosely-timed coding style. Temporal decoupling permits a significant simulation speed improvement by reducing the number of context switches and events.
 - Abstraction trades off accuracy for higher simulation speed
- Problem: PDES is a different foundation than DES
 - TD design assumptions are not necessarily true for PDES
 - Global time quantum is a technical obstacle (race condition)
- Proposal: Reevaluate costs/benefits, redesign if needed
 - Analyze TD idea for PDES, adopt advantages, drop drawbacks
 - Avoid `t1m_global_quantum`, promote `wait(time)`
 - Consider the use of a compiler to optimize scheduling, timing
 - Out-of-Order PDES is one solution (fully automatic, accurate)

Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

39

Overcoming the Obstacles

- Obstacle 7:
Ongoing...
- Investigation
needs
examples
- Speed vs.
accuracy
tradeoff
in PDES
 - Out-of-order
PDES can
likely achieve the same benefit
 - Without loss of accuracy (?)

Obstacle 7: Temporal Decoupling

- Fact: TD is designed to speed up sequential DES
 - IEEE 1666-2011, Section 12.1 on "TLM-2.0 global quantum" (page 453):
Temporal decoupling permits SystemC processes to run ahead of simulation time for an amount of time known as the time quantum and is associated with the loosely-timed coding style. Temporal decoupling permits a significant simulation speed improvement by reducing the number of context switches and events.
 - Abstraction trades off accuracy for higher simulation speed
- Problem: PDES is a different foundation than DES
 - TD design assumptions are not necessarily true for PDES
 - Global time quantum is a technical obstacle (race condition)
- Proposal: Reevaluate costs/benefits, redesign if needed
 - Analyze TD idea for PDES, adopt advantages, drop drawbacks
 - Avoid `tlm_global_quantum`, promote `wait(time)`
 - Consider the use of a compiler to optimize scheduling, timing
 - Out-of-Order PDES is one solution (fully automatic, accurate)

Seven Obstacles in the Way of Parallel SystemC Simulation (c) 2016 R. Doemer, CECS 15

Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS 40

Summary and Analysis

- Overcoming 7 Obstacles towards Parallel SystemC
 1. Co-Routine Semantics: *Resolved*
 2. Simulator State: *Ongoing...*
 3. Lack of Thread Safety: *Ongoing...*
 4. Class `sc_channel`: *Fixed*
 5. TLM-2.0: *Reevaluated, Resolved*
 6. Sequential Mindset: *Not a problem*
 7. Temporal Decoupling: *Ongoing...*
- So the problem is not solved yet,
but we're getting closer!
- Next, let's look at the state of the art:
Recoding Infrastructure for SystemC (RISC)

Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS 41

Poll: SystemC Future

- Towards truly parallel simulation, do you expect the SystemC standard to further evolve?
[Single Choice]
 - Answer 1: Yes, SystemC will evolve.
 - Answer 2: No, SystemC will remain as is.
 - Answer 3: No, SystemC will be replaced with another language.
 - Answer 4: Not sure, I don't know.

Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

42

Out-of-Order Parallel Simulation of SystemC Models using the RISC Framework

Part 3: RISC: Recoding Infrastructure for SystemC

Rainer Dömer
doemer@uci.edu

Center for Embedded and Cyber-Physical Systems
University of California, Irvine

UCIrvine
University of California, Irvine



Recoding Infrastructure for SystemC (RISC)

- Advanced Parallel SystemC Simulation
 - Aggressive PDES on many-core host platforms
 - Maximum compliance with IEEE SystemC semantics
- Introduction of a Dedicated SystemC Compiler
 - Advanced conflict analysis for safe parallel execution
 - Automatic model instrumentation and code generation
- Parallel SystemC Simulator
 - Out-of-order parallel scheduler, multi-thread safe primitives
 - Multi- and many-core host platforms (e.g. Intel® Xeon Phi™)
- Open Source
 - Freely available for evaluation and collaboration
 - BSD license
 - Thanks to Intel Corporation!

Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

44

Poll: Simulator Run Time

- For your larger SystemC design models, how long is the typical simulator run time?
[Single Choice]
 - Answer 1: A few seconds
 - Answer 2: About a minute
 - Answer 3: About 10 minutes
 - Answer 4: About an hour
 - Answer 5: Several hours
 - Answer 6: About a day
 - Answer 7: Several days.
 - Answer 8: Not sure, I don't know.

Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

45

Recoding Infrastructure for SystemC (RISC)

- Out-of-Order PDES Key Ideas
 1. Dedicated *SystemC compiler* with advanced model analysis
 - Static conflict analysis based on Segment Graphs
 2. *Parallel simulator* with out-of-order scheduling
 - Fast decision making at run-time, optimized mapping
- Fundamental Data Structure: *Segment Graph*
 - Key to semantics-compliant out-of-order execution [DATE'12]
 - Key to prediction of future thread state [DATE'13]
 - “Optimized Out-of-Order Parallel DE Simulation Using Predictions”
 - Key to May-Happen-in-Parallel Analysis [DATE'14]
 - “May-Happen-in-Parallel Analysis based on Segment Graphs for Safe ESL Models” (**Best Paper Award**)
 - Combined: “OoO PDES for TLM” [IEEE TCAD'14]
 - Comprehensive summary with HybridThreads extension

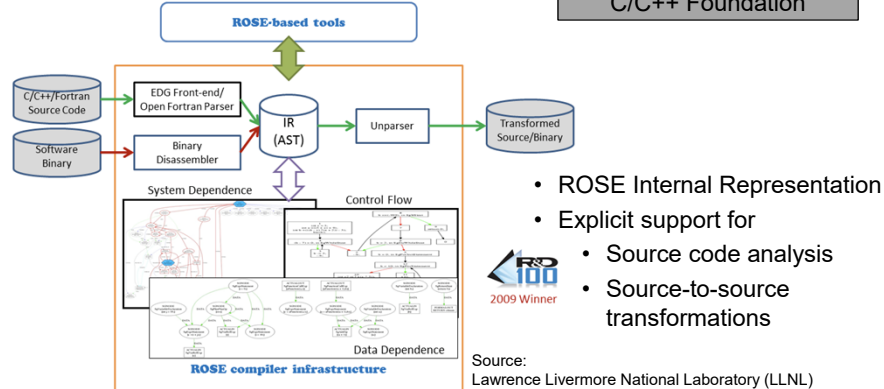
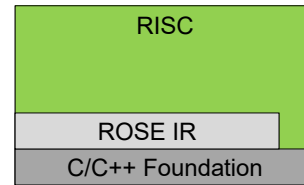
Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

46

Recoding Infrastructure for SystemC (RISC)

- RISC Software Stack
 - *Recoding Infrastructure for SystemC*
 - C/C++ foundation
 - ROSE compiler (from LLNL)



- ROSE Internal Representation
- Explicit support for
 - Source code analysis
 - Source-to-source transformations

Source: Lawrence Livermore National Laboratory (LLNL)

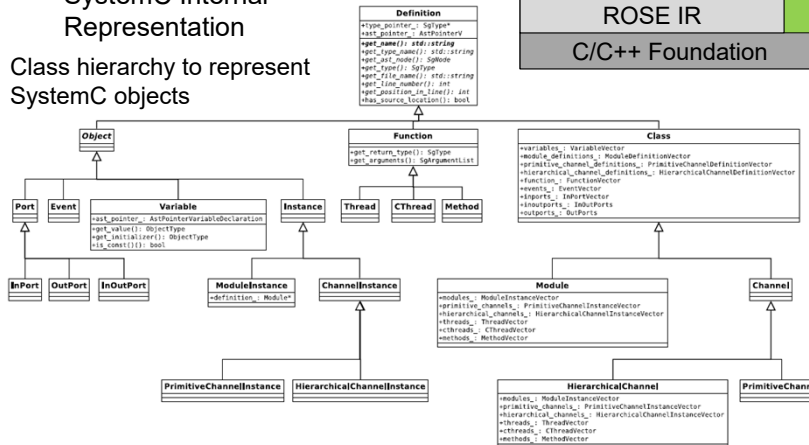
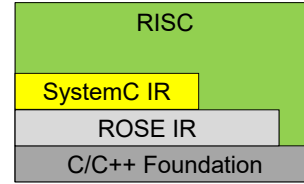
Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

47

Recoding Infrastructure for SystemC (RISC)

- RISC Software Stack
 - *Recoding Infrastructure for SystemC*
 - SystemC Internal Representation
- Class hierarchy to represent SystemC objects

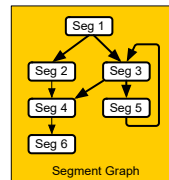
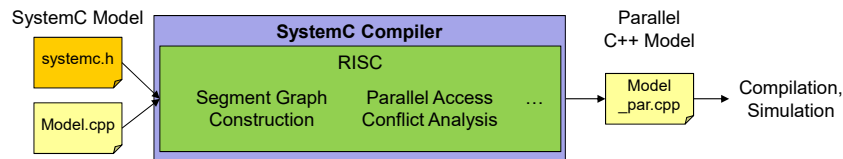
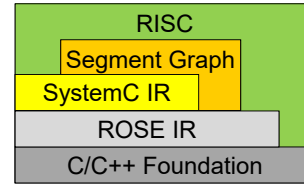


Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS 48

RISC Compiler

- RISC Software Stack
 - *Recoding Infrastructure for SystemC*
 - 1) Segment Graph construction
 - 2) Parallel access conflict analysis



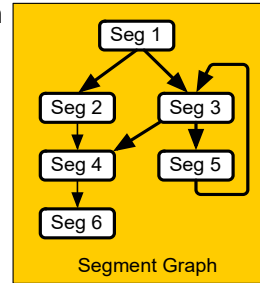
Step 1: Build a Segment Graph (SG)

Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS 49

RISC Compiler

- Segment Graph Construction
 - Segment Graph (SG) is a directed graph
 - Nodes: Segments
 - Code statements executed between two scheduling steps
 - Expression statements
 - Control flow statements (if, while, ...)
 - Function calls
 - Edges: Segment boundaries
 - Primitives that trigger scheduler entry
 - wait(event)
 - wait(time)
 - Segment Graph can be constructed statically by the compiler from the model source code
 - Let's look into this in detail by use of a few examples!



Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS 50

RISC Compiler

- Segment Graph Construction
 - Example: Source code and Segment Graph

```

int a;
if(cond) {
    int b;
    wait(1);
    int c;
} else {
    int d;
}
int e;
wait(2);
int f;
while(cond) {
    int g;
}
int h;
```

Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS 51

RISC Compiler

- Segment Graph Construction
 - Example for straight-line code

```

void straight()
{
    x = 42;
    int xx = 43;
    int yy;
    YY;
    int o = y;

    wait(10, SC_NS);

    wait();

    int kk;

    wait();

    int oo;
}
    
```

```

graph TD
    S0["Segment ID: 0  
input_straight.cpp:24 (this) -> x = 42  
input_straight.cpp:25 int xx = 43;  
input_straight.cpp:26 int yy;  
input_straight.cpp:27 yy  
input_straight.cpp:28 int o =(this) -> y;"]
    S1["Segment ID: 1 (input_straight.cpp:30)"]
    S2["Segment ID: 2 (input_straight.cpp:32)  
input_straight.cpp:34 int kk;"]
    S3["Segment ID: 3 (input_straight.cpp:37)  
input_straight.cpp:39 int oo;"]
    S0 --> S1
    S1 --> S2
    S2 --> S3
    
```

Tutorial at ESWEEK, Sept. 20, 2020 (c) 2020 R. Doemer et al., CECS 52

RISC Compiler

- Segment Graph Construction
 - Example for conditional control flow
 - if, if-else, switch-case (with break)

```

void if_statement()
{
    wait();
    int aaa;
    if(test) {
        int bbb;
        wait();
        int ccc;
    }
    int ddd;
    wait();
    int eee;
}
    
```

```

graph TD
    S0["Segment ID: 0  
input_if_else.cpp:27  
input_if_else.cpp:28 int aaa;  
compilerGenerated:0 (this) -> test  
input_if_else.cpp:30 int bbb;  
input_if_else.cpp:34 int ddd;"]
    S1["Segment ID: 1 (input_if_else.cpp:31)  
input_if_else.cpp:32 int ccc;  
input_if_else.cpp:34 int ddd;"]
    S2["Segment ID: 2 (input_if_else.cpp:35)  
input_if_else.cpp:36 int eee;"]
    S0 --> S1
    S0 --> S2
    S1 --> S2
    
```

Tutorial at ESWEEK, Sept. 20, 2020 (c) 2020 R. Doemer et al., CECS 53

RISC Compiler

- Segment Graph Construction
 - Example for repetition (loops)
 - while, do-while, for (with break, continue)

```

void while_continue_statement()
{
    int kk;
    while(test){
        int aa;
        wait();
        int bb;
        if(test1) {
            continue;
        }
        int oo;
        wait();
        int cc;
    }
    int dd;
    wait();
}
    
```

Tutorial at ESWEEK, Sept. 20, 2020
(c) 2020 R. Doemer et al., CECS 54

RISC Compiler

- Segment Graph Construction
 - Example for function calls
 - f(x), return

```

void f()    int g1()
{
    int aa;    int g_0;
    wait();    wait();
    int bb;    int g_1 = 33;
    g1();      if(g_1 == 88) {
    int cc;      int g_2;
    wait();      wait();
    int dd;      int g_3 = 44;
                return 43;
                int DEAD_CODE;
            }
            int g_4;
            wait();
            int g_5;
            wait();
            int g_6;
            int return_value = 2;
            return return_value;
        }
    
```

Tutorial at ESWEEK, Sept. 20, 2020
(c) 2020 R. Doemer et al., CECS 55

RISC Compiler

- Segment Graph Construction
 - Example for recursive function calls
 - Direct, indirect recursion

```

void main()
{ wait();
  f();
  wait();
}

void g()
{ xx--;
  wait();
  if(xx>0) {
    wait();
    int before_rec;
    f();
    int after_rec;
    wait();
  } else {
    wait();
    return;
  }
}

void f()
{ wait();
  if(xx>0) {
    wait();
    g();
    wait();
  }
  return;
}
    
```

Tutorial at ESWEEK, Sept. 20, 2020 (c) 2020 R. Doemer et al., CECS 56

RISC Compiler

- Parallel Access Conflict Analysis for Segments
 - Need to comply with SystemC LRM [IEEE Std. 1666™]
 - Cooperative (or co-routine) multitasking semantics
 - “process instances execute without interruption”
 - System designer “can assume that a method process will execute in its entirety without interruption”
 - A parallel implementation “would be obliged to analyze any dependencies between processes and constrain their execution to match the co-routine semantics.”
 - Must avoid race conditions when using shared variables!
 - Prevent conflicting segments to be scheduled in parallel

Conflict	Seg 1	Seg 2	Seg 3
Seg 1	True		
Seg 2		True	True
Seg 3		True	

Seg 2

R: a, b

W: x

RW: z

Seg 3

R: a, b

W: x, y

RW:

Tutorial at ESWEEK, Sept. 20, 2020 (c) 2020 R. Doemer et al., CECS 57

RISC Compiler

- Parallel Access Conflict Analysis for Segments
 - Variable analysis for Read, Write, and Read/Write accesses
 - Example:

```

class Conflict: public sc_module {
  SC_CTOR(Conflict)
  { SC_THREAD(thread1);
    SC_THREAD(thread2);
  }
  int x, y, z;

  void thread1()
  {
    int a;
    a = 2;
    wait();
    a = x + y;
    wait();
    z++;
  };

  void thread2()
  {
    int b = 2;
    x = y;
    wait();
    x = y * z;
    wait();
    z++;
    wait();
    x++;
  }
    
```

Segment ID: 0

conflict.cpp:24 int a;

conflict.cpp:25 a = 2

Segment ID: 3

conflict.cpp:34 int b = 2;

conflict.cpp:35 x = y

Segment ID: 1 (conflict.cpp:26)

conflict.cpp:27 a = x + y

Segment ID: 4 (conflict.cpp:36)

conflict.cpp:37 x = y * z

Segment ID: 2 (conflict.cpp:28)

conflict.cpp:29 z++

Segment ID: 5 (conflict.cpp:38)

conflict.cpp:39 z++

Segment ID: 6 (conflict.cpp:40)

conflict.cpp:41 x++

Segment Graph

Tutorial at ESWEEK, Sept. 20, 2020 (c) 2020 R. Doemer et al., CECS 58

RISC Compiler

- Parallel Access Conflict Analysis for Segments
 - Variable analysis for Read, Write, and Read/Write accesses
 - Example:

Segment ID: 0

(W) a

Segment ID: 3

(W) x

(W) b

(R) y

Segment ID: 1 (conflict.cpp:26)

(W) a

(R) x

(R) y

Segment ID: 4 (conflict.cpp:36)

(W) x

(R) y

(R) z

Segment ID: 2 (conflict.cpp:28)

(W) z

(R) z

Segment ID: 5 (conflict.cpp:38)

(W) z

(R) z

Segment ID: 6 (conflict.cpp:40)

(W) x

(R) x

Variable Accesses

Segment ID: 0

conflict.cpp:24 int a;

conflict.cpp:25 a = 2

Segment ID: 3

conflict.cpp:34 int b = 2;

conflict.cpp:35 x = y

Segment ID: 1 (conflict.cpp:26)

conflict.cpp:27 a = x + y

Segment ID: 4 (conflict.cpp:36)

conflict.cpp:37 x = y * z

Segment ID: 2 (conflict.cpp:28)

conflict.cpp:29 z++

Segment ID: 5 (conflict.cpp:38)

conflict.cpp:39 z++

Segment ID: 6 (conflict.cpp:40)

conflict.cpp:41 x++

Segment Graph

Tutorial at ESWEEK, Sept. 20, 2020 (c) 2020 R. Doemer et al., CECS 59

RISC Compiler

- Parallel Access Conflict Analysis for Segments
 - Variable analysis for Read, Write, and Read/Write accesses
 - Example:

Variable Accesses

Segment ID: 0	Segment ID: 3
(W) a	(W) x
(R) x	(W) b
(R) y	(R) y
Segment ID: 1 (conflict.cpp:26)	Segment ID: 4 (conflict.cpp:36)
(W) a	(W) x
(R) x	(R) y
(R) y	(R) z
Segment ID: 2 (conflict.cpp:28)	Segment ID: 5 (conflict.cpp:38)
(W) z	(W) z
(R) z	(R) z
Segment ID: 6 (conflict.cpp:40)	
(W) x	
(R) x	

	0	1	2	3	4	5	6
0							
1				X			
2							
3							
4		X					
5							
6							

Data Conflict Table

Tutorial at ESWEEK, Sept. 20, 2020
(c) 2020 R. Doemer et al., CECS
60

RISC Compiler

- RISC Software Stack
 - Recoding Infrastructure for SystemC
 - Segment Graph construction
 - Parallel access conflict analysis

SystemC Model

systemc.h

Model.cpp

SystemC Compiler

RISC

Segment Graph Construction

Parallel Access Conflict Analysis

Parallel C++ Model

Model_par.cpp

Compilation, Simulation

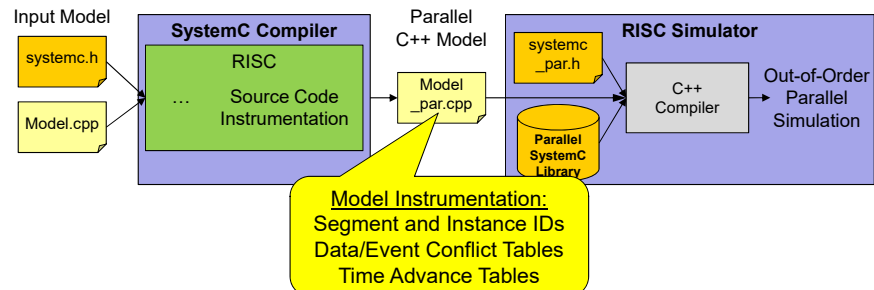
Instrumentation!

	Seg 1	Seg 2	Seg 3
Conflict			
Seg 1	True		
Seg 2		True	True
Seg 3		True	

Tutorial at ESWEEK, Sept. 20, 2020
(c) 2020 R. Doemer et al., CECS
61

SystemC Compiler and Simulator

- Compiler and Simulator work hand in hand
 - Compiler performs conservative static analysis
 - Analysis results are passed to the simulator
 - Simulator can make safe scheduling decisions quickly
- Automatic Model Instrumentation
 - Static analysis results are inserted into the source code



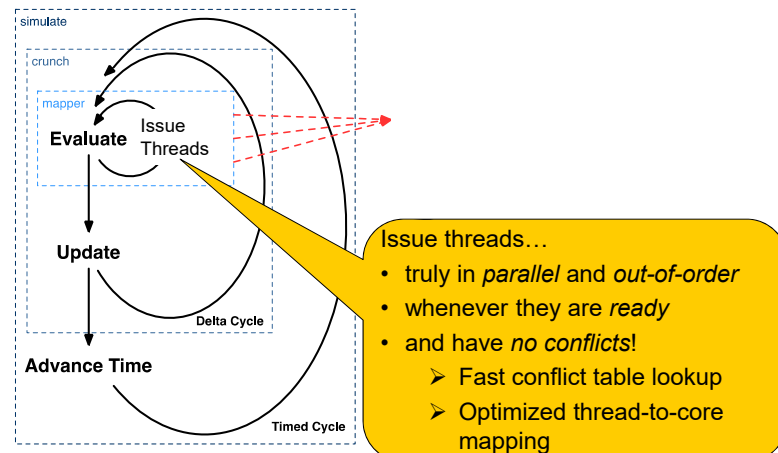
Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

62

RISC Simulator

- Simulator Kernel with Out-of-Order Parallel Scheduler
 - Conceptual OoO PDES execution



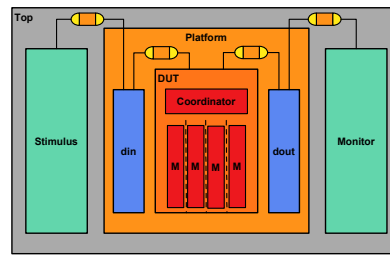
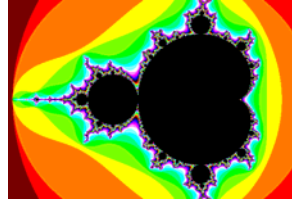
Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

63

Experiments and Results

- Mandelbrot Renderer (Graphics Pipeline Application)
 - Mandelbrot Set
 - Mathematical set of points in complex plane
 - Two-dimensional fractal shape
 - High computation load
 - Recursive/iterative function
 - Embarassingly parallel
 - Parallelism at pixel level
 - SystemC Model
 - TLM abstraction
 - Horizontal image slices
 - Highly configurable
 - Parallelism parameter from 1 to 256 slices



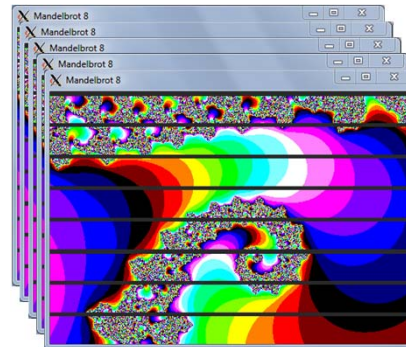
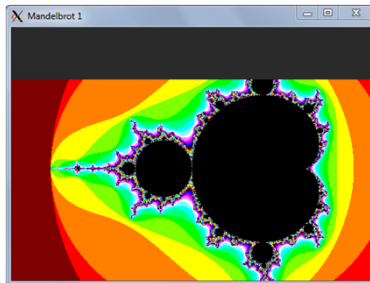
Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

64

Experiments and Results

- Mandelbrot Renderer (Graphics Pipeline Application)
 - *Simulated Graphics Demonstration*
(when network delays prevent actual graphical demo)



Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

65

Experiments and Results

- Mandelbrot Renderer (Graphics Pipeline Application)
 - Simulator run times on 16-core Intel® Xeon® multi-core host
 - 2 CPUs at 2.7 GHz, 8 cores each, 2-way hyper-threaded
 - RISC V0.2.1, Posix-threads

Parallel Slices	DES		PDES			OOO PDES		
	Run Time	CPU Load	Run Time	CPU Load	Speedup	Run Time	CPU Load	Speedup
1	162.13 s	99%	162.06 s	100%	1.00 x	161.90 s	100%	1.00 x
2	162.19 s	99%	96.50 s	168%	1.68 x	96.48 s	168%	1.68 x
4	162.56 s	99%	54.00 s	305%	3.01 x	53.85 s	304%	3.02 x
8	163.10 s	99%	29.89 s	592%	5.46 x	30.05 s	589%	5.43 x
16	164.01 s	99%	19.03 s	1050%	8.62 x	20.08 s	997%	8.17 x
32	165.89 s	99%	11.78 s	2082%	14.08 x	11.99 s	2023%	13.84 x
64	170.32 s	99%	9.79 s	2607%	17.40 x	9.85 s	2608%	17.29 x
128	174.55 s	99%	9.34 s	2793%	18.69 x	9.39 s	2787%	18.59 x
256	185.47 s	100%	8.91 s	2958%	20.82 x	8.90 s	2964%	20.84 x

Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

66

Experiments and Results

- Many-Core Target Platform: Intel® Xeon Phi™
 - Many Integrated Core (MIC) architecture
 - 1 Coprocessor 5110P CPU at 1.052 GHz
 - 60 physical cores with 4-way hyper-threading
 - Appears as regular Linux host with 240 cores
 - Up to 8 lanes available for vector processing
- RISC extended for exploiting 2 types of parallelism
 - Out-of-Order PDES: thread-level parallelism
 - Intel® compiler SIMD: data-level parallelism
 - RISC SIMD Advisor identifies functions with data-level parallelism suitable for SIMD vectorization
 - DAC '17 paper:
 - “Exploiting Thread and Data Level Parallelism for Ultimate Parallel SystemC Simulation”



Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

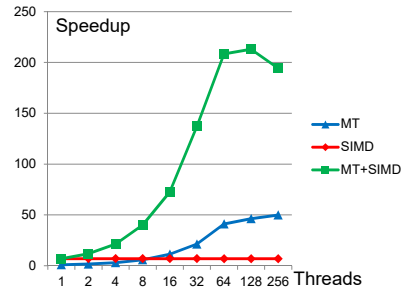
67

Experiments and Results

- Many-Core Target Platform: Intel® Xeon Phi™
 - Exploiting thread- and data-level parallelism [DAC'17]
 - Mandelbrot renderer (graphics pipeline application)

- Experimental Results:

PAR	MT	SIMD	MT+SIMD
1	1.00	6.92	6.94
2	1.68	6.92	11.77
4	3.04	6.92	21.19
8	5.84	6.92	40.10
16	11.37	6.92	72.52
32	21.32	6.91	137.21
64	41.07	6.90	208.41
128	46.29	6.89	212.96
256	49.90	6.87	194.19



- Increasing degree of parallelism (PAR = number of threads) reaches a combined multi-threading (MT) and data-level (SIMD) speedup of **up to 212x!**

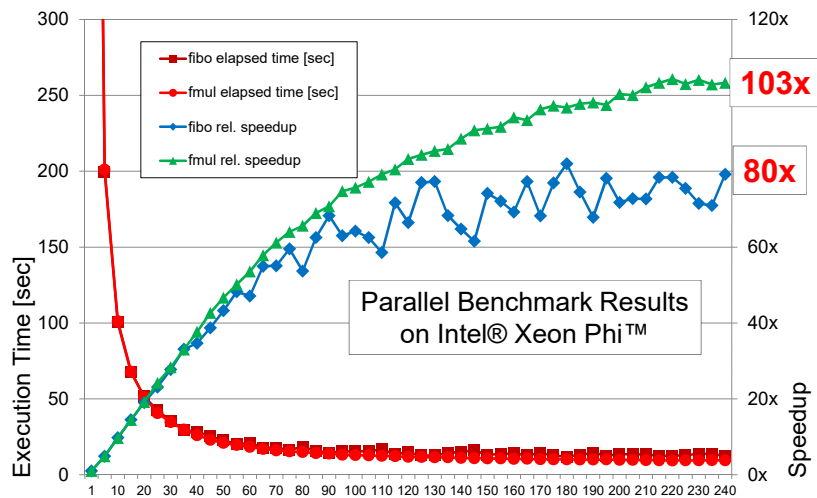
Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

68

Experiments and Results

- Parallel Benchmark Results (Xeon Phi Coprocessor, 60x4 cores)



Parallel Benchmark Results on Intel® Xeon Phi™

Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

69

Conclusion

- Recoding Infrastructure for SystemC (RISC)
- Out-of-Order Parallel SystemC Simulation
 - Aggressive PDES on many-core host platforms
 - Maximum compliance with IEEE SystemC semantics
- Introduction of a Dedicated SystemC Compiler
 - Advanced conflict analysis for safe parallel execution
 - Automatic model instrumentation and code generation
- Parallel SystemC Simulator
 - Out-of-order parallel scheduler, multi-thread safe primitives
 - Multi- and many-core host platforms (e.g. Intel® Xeon Phi™)
- Open Source
 - Freely available for use and collaboration (BSD license)
 - Thanks to Intel Corporation!

Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

70

Acknowledgments

- For solid work, fruitful discussions, and honest feedback, I would like to thank:
 - My team at UCI
 - Emad Arasteh, Aditya Harit, Vivek Govindasamy
 - Zhongqi Cheng, Daniel Mendoza
 - Tim Schmidt, Guantao Liu
 - Farah Arabi, Spencer Kam
 - Our collaborators at Intel
 - Ajit Dingankar
 - Desmond Kirkpatrick
 - Abhijit Davare
 - Philipp Hartmann
 - And many others...
- This work has been supported in part by substantial funding from Intel Corporation. Thank you!

Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

71

Out-of-Order Parallel Simulation of SystemC Models using the RISC Framework

Part 4: Hands-on Practical Training with RISC Compiler and Simulator

Rainer Dömer
doemer@uci.edu

Center for Embedded and Cyber-Physical Systems
University of California, Irvine



RISC Open Source Releases

- [RISCv062] Z. Cheng, R. Dömer, A. Harit.
RISC Compiler and Simulator, Release V0.6.2. September 2020.
– <http://cecs.uci.edu/~doemer/risc.html#RISC062>
- [RISCv060] Z. Cheng, R. Dömer, S. Kam, and D. Mendoza.
RISC Compiler and Simulator, Release V0.6.0. September 2019.
- [RISCv050] Z. Cheng, R. Dömer, D. Mendoza, and T. Schmidt.
RISC Compiler and Simulator, Release V0.5.0. September 2018.
- [RISCv042] Z. Cheng, R. Dömer, D. Mendoza, and T. Schmidt.
RISC Compiler and Simulator, Release V0.4.2. June 2018.
- [RISCv040] R. Dömer, G. Liu, and T. Schmidt.
RISC Compiler and Simulator, Release V0.4.0. July 2017.
- [RISCv030] R. Dömer, G. Liu, and T. Schmidt.
RISC Compiler and Simulator, Beta Release V0.3.0. September 2016.
- [RISCv021] R. Dömer, G. Liu, and T. Schmidt.
RISC Compiler and Simulator, Alpha Release V0.2.1. October 2015.
- [RISCv020] R. Dömer, G. Liu, and T. Schmidt.
RISC Compiler and Simulator, Alpha Release V0.2.0. September 2015.
- [RISCv010] R. Dömer, G. Liu, and T. Schmidt.
RISC API, Alpha Release V0.1.0. June 2014.

RISC Open Source Releases

- RISC Compiler and Simulator, Release V0.6.2
 - <http://www.cecs.uci.edu/~doemer/risc.html#RISC062>
 - Installation notes and script: **INSTALL, Makefile**
 - Open source tar ball: **risc_v0.6.2.tar.gz**
 - Docker script and container: **Dockerfile**
 - Doxygen documentation: **RISC API, OOPSC API**
 - Tool manual pages: **risc, visual, sysdot ...**
 - BSD license terms: **LICENSE**
 - Companion Technical Report
 - CECS Technical Report 19-04: **CECS_TR_19_04.pdf**

```
bash# docker pull ucirvinelecs/risc062
bash# docker run -it ucirvinelecs/risc062
[dockeruser]# cd demodir
[dockeruser]# make play_demo
```

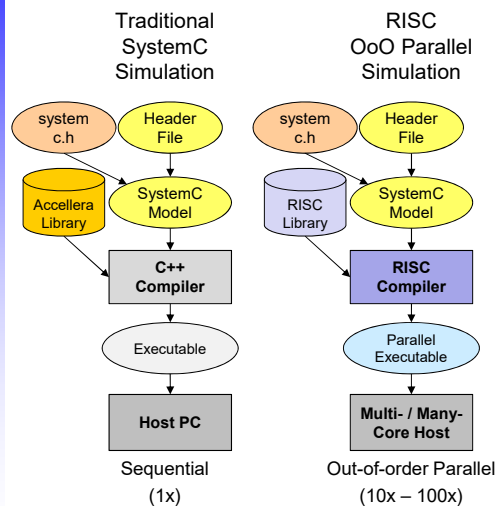
- Docker container: <https://hub.docker.com/r/ucirvinelecs/risc062/>

Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

74

RISC Compiler Tool Flow



- Compile and simulate with Accellera:
 - `g++ play.cc ...`
 - `./play_seq`
- Compile and simulate with RISC:
 - `risc play.cc ...`
 - `./play_ooo`
- Measure simulator run time:
 - `/usr/bin/time ...`

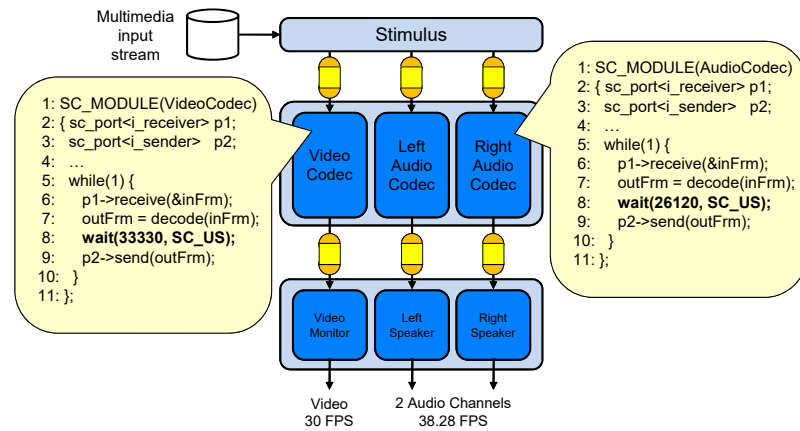
Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

75

Demo Example 1

- Conceptual DVD Player, TLM-1.0 style
 - Parallel video and audio decoding with different frame rates



Tutorial at ESWEEK, Sept. 20, 2020

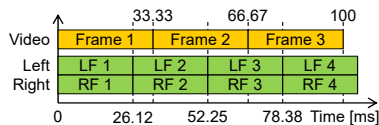
(c) 2020 R. Doemer et al., CECS

76

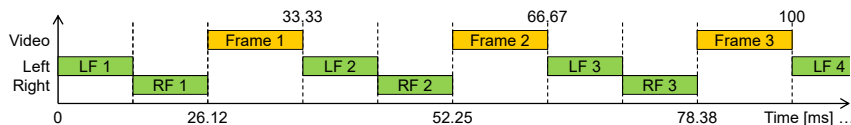
Demo Example 1

- Conceptual DVD Player, TLM-1.0 style
 - Parallel video and audio decoding with different frame rates

1. Real time schedule: fully parallel



2. Reference simulator schedule (DES)



Tutorial at ESWEEK, Sept. 20, 2020

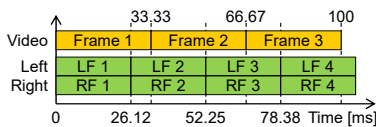
(c) 2020 R. Doemer et al., CECS

77

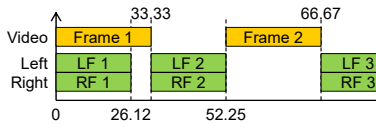
Demo Example 1

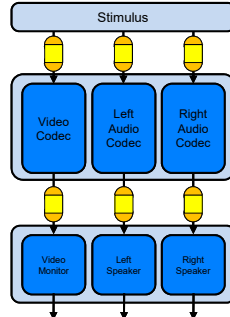
- Conceptual DVD Player, TLM-1.0 style
 - Parallel video and audio decoding with different frame rates

1. Real time schedule: fully parallel



3. Synchronous parallel schedule (PDES)



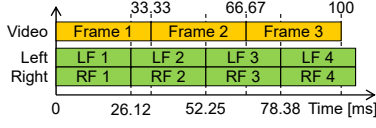


Tutorial at ESWEEK, Sept. 20, 2020 (c) 2020 R. Doemer et al., CECS 78

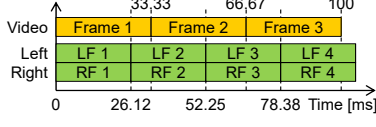
Demo Example 1

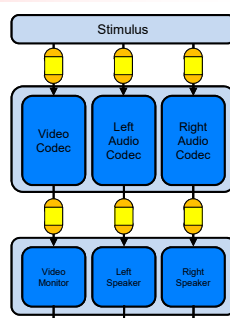
- Conceptual DVD Player, TLM-1.0 style
 - Parallel video and audio decoding with different frame rates

1. Real time schedule: fully parallel



4. Out-of-order parallel schedule (OoO PDES)



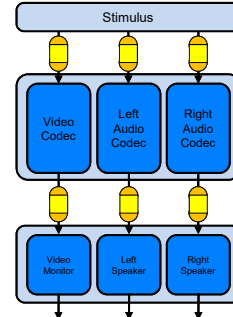


Tutorial at ESWEEK, Sept. 20, 2020 (c) 2020 R. Doemer et al., CECS 79

Demo Example 1

- Conceptual DVD Player, TLM-1.0 style
 - Parallel video and audio decoding with different frame rates
- Simulator Run Times
 - 4-core Intel® Xeon® CPU at 3.4 GHz
 - RISC v0.2.1, Posix-threads

		DES	PDES	OoO PDES
10 sec stream	Run Time	6.98 s	4.67 s	2.94 s
	CPU Load	97%	145%	238%
	Speedup	1 x	1.49 x	2.37 x
100 sec stream	Run Time	68.21 s	45.91 s	28.13 s
	CPU Load	100%	149%	251%
	Speedup	1 x	1.49 x	2.42 x



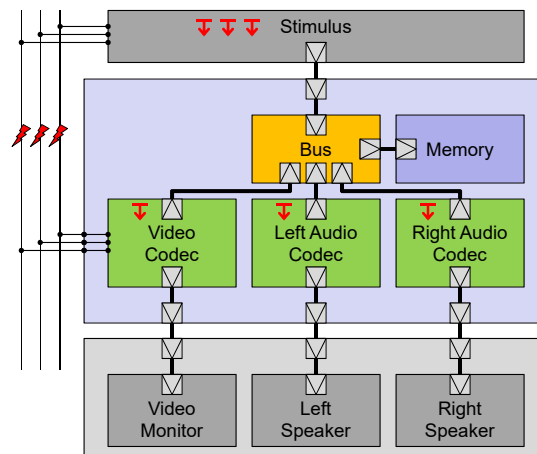
Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

80

Demo Example 2

- Conceptual DVD Player, TLM-2.0 style
 - Example: hierarchical socket binding, event handshakes



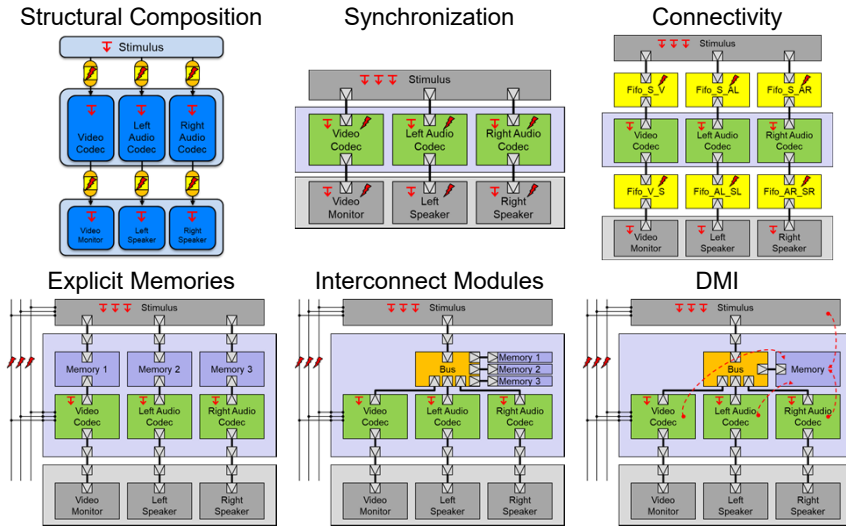
Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

81

Demo Example 2

- Various Modeling Styles Supported by RISC v0.6.2



Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

82

Demo Example 2

- Experimental Results for TLM-2.0 DVD Player Models
 - All models are functional and simulate correctly (RISC v0.6.0)
 - Results: run time (seconds) and speedup (%)

Interface	Direct			Hierarchical			Interconnect		
	Seq	OoO Par	282%	Seq	OoO Par	274%	Seq	OoO Par	278%
BTI	208.1	73.8	282%	208.1	75.7	274%	208.4	74.8	278%
DMI	208.2	73.7	282%	208.5	75.5	276%	208.4	74.7	279%
NBTI	209.3	74.9	279%	209.4	75.6	277%	209.5	75.7	277%

- All models exhibit high simulation speedup
 - 2.8 times faster than the sequential reference model
 - This beats our prior results: TLM-1.0 reached only 2.5 x

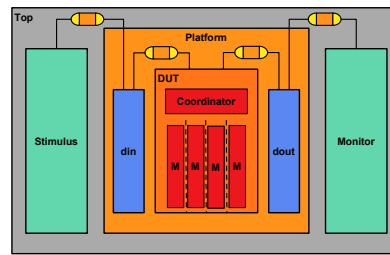
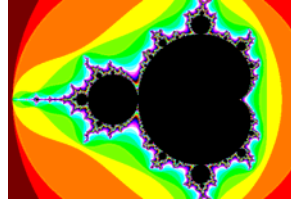
Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

83

Demo Example 3

- Mandelbrot Renderer (Graphics Pipeline Application)
 - Mandelbrot Set
 - Mathematical set of points in complex plane
 - Two-dimensional fractal shape
 - High computation load
 - Recursive/iterative function
 - Embarrassingly parallel
 - Parallelism at pixel level
 - SystemC Model
 - TLM abstraction
 - Horizontal image slices
 - Highly configurable
 - Parallelism parameter from 1 to 256 slices



Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

84

Demo Example 3

- Experimental Results
 - Simulator run times on 16-core Intel® Xeon® multi-core host
 - 2 CPUs at 2.7 GHz, 8 cores each, 2-way hyper-threaded
 - RISC V0.2.1, Posix-threads

Parallel Slices	DES		PDES			OOO PDES		
	Run Time	CPU Load	Run Time	CPU Load	Speedup	Run Time	CPU Load	Speedup
1	162.13 s	99%	162.06 s	100%	1.00 x	161.90 s	100%	1.00 x
2	162.19 s	99%	96.50 s	168%	1.68 x	96.48 s	168%	1.68 x
4	162.56 s	99%	54.00 s	305%	3.01 x	53.85 s	304%	3.02 x
8	163.10 s	99%	29.89 s	592%	5.46 x	30.05 s	589%	5.43 x
16	164.01 s	99%	19.03 s	1050%	8.62 x	20.08 s	997%	8.17 x
32	165.89 s	99%	11.78 s	2082%	14.08 x	11.99 s	2023%	13.84 x
64	170.32 s	99%	9.79 s	2607%	17.40 x	9.85 s	2608%	17.29 x
128	174.55 s	99%	9.34 s	2793%	18.69 x	9.39 s	2787%	18.59 x
256	185.47 s	100%	8.91 s	2958%	20.82 x	8.90 s	2964%	20.84 x

Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

85

Interactive Hands-on Demo

- Docker Setup
 - `sudo docker pull ucirvinelecs/risc062`
 - `sudo docker run -it ucirvinelecs/risc062:latest`
 - `cd demodir`
- Demo 1: DVD Player, TLM-1.0
 - `make play_demo`
- Demo 2: DVD Player, TLM-2.0
 - `make play_TLM2_bus_mem_demo`
- Demo 3: Mandelbrot Renderer
 - `make mandelbrot_demo`
- Handout and detailed “Cheat Sheet” available online
 - <http://www.cecs.uci.edu/~doemer/ESWeekTutorial.pdf>
 - <http://www.cecs.uci.edu/~doemer/ESWeekTutorial.txt>

Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

86

Out-of-Order Parallel Simulation of SystemC Models using the RISC Framework

Part 5: Hands-on Practical Analysis of Parallel Potential of SystemC Models

Rainer Dömer
doemer@uci.edu

Center for Embedded and Cyber-Physical Systems
University of California, Irvine

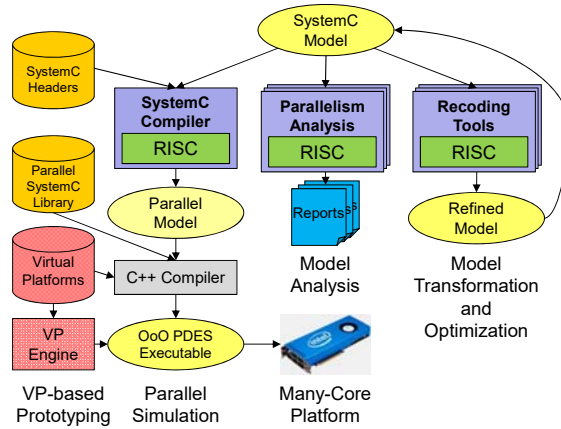
UCIrvine
University of California, Irvine



RISC Framework Overview

- RISC Framework consists of 3 Branches

- Simulation
 - With VP support
 - Analysis
 - Static
 - Dynamic
 - Recoding
 - Transformation
 - Optimization
- This session demonstrates SystemC analysis features

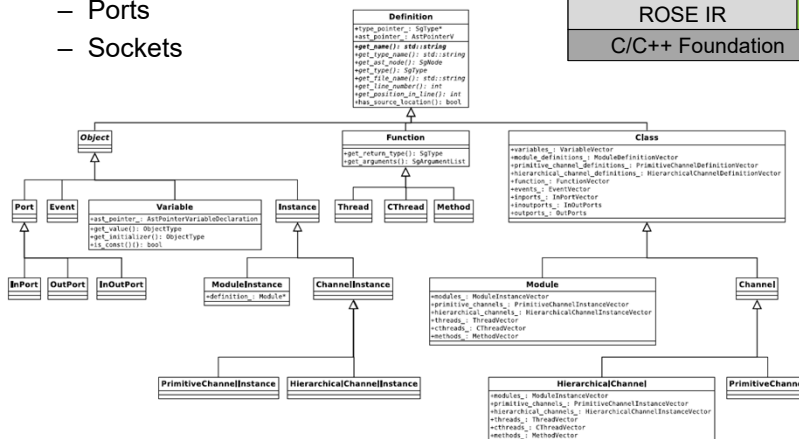
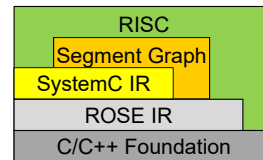


Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS 88

Analysis of Model Structure

- SystemC Model Hierarchy
- SystemC Model Connectivity
 - Ports
 - Sockets

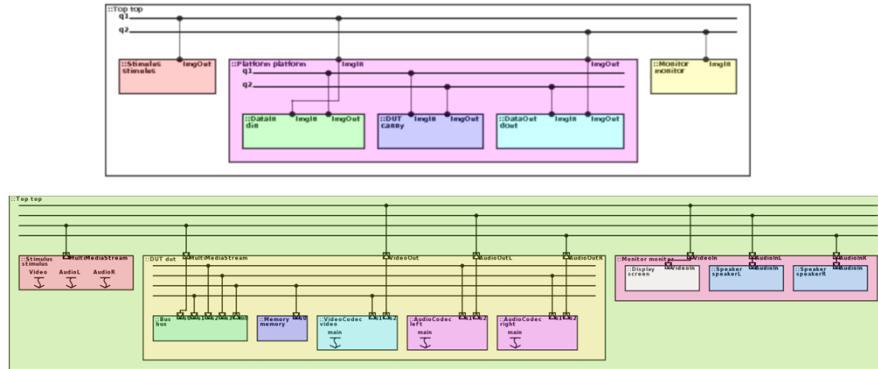


Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS 89

Visualization of Model Structure

- SystemC Model Visualization: `visual`
 - Hierarchy and connectivity
 - Ports and sockets
 - Threads in modules



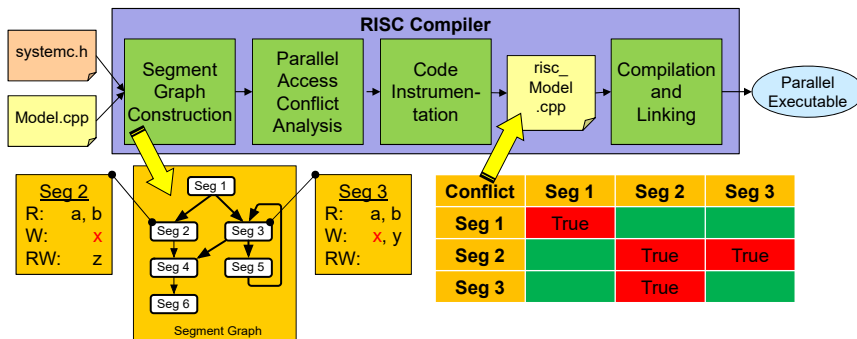
Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

90

Analysis of Potential Parallelism

- Segment Graph based Conflict Analysis
 1. Build the Segment Graph
 2. Perform parallel access conflict analysis
 3. Instrument the model for parallel execution



Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

91

Visualization of Segment Graph

- Segment Graph and Conflicts Visualization: **sysdot**

```

mandelbrot.cpp:899 iteration = iteration + 1
mandelbrot.cpp:899 color =(iteration % 16)
mandelbrot.cpp:899 (this) -> image . R[512 - 1 - row][col] = palette[color][0]
mandelbrot.cpp:899 (this) -> image . G[512 - 1 - row][col] = palette[color][1]
mandelbrot.cpp:899 (this) -> image . B[512 - 1 - row][col] = palette[color][2]
mandelbrot.cpp:899 col++

```

Tutorial at ESWEEK, Sept. 20, 2020 (c) 2020 R. Doemer et al., CECS 92

Visualization of Conflict Tables

- Web browser and **sysdot**

:t	Coordinates::t	Coordinates::t	Coordinates::t
:l	Image::R	Image::R	Image::R
:r	Image::G	Image::G	Image::G
:b	Image::B	Image::B	Image::B
:t	Coordinates::t	Coordinates::t	Coordinates::t
:l	Image::R	Image::R	Image::R
:r	Image::G	Image::G	Image::G
:b	Image::B	Image::B	Image::B
:t	Coordinates::t	Coordinates::t	Coordinates::t

Tutorial at ESWEEK, Sept. 20, 2020 (c) 2020 R. Doemer et al., CECS 93

Interactive Hands-on Demo

- Docker Setup
 - `xhost +`
 - `sudo docker run -it --net=host --env="DISPLAY" \`
 - `-volume="$HOME/.Xauthority:/root/.Xauthority:rw" \`
 - `ucirvinelecs/risc062:latest`
 - `cd demodir`
- Demo 5: Examples using `visual`
 - `visual play.cpp` (and other examples)
- Demo 6: Examples using `sysdot`
 - `make play_ooo`
 - `sysdot play_segment_graph.dot` (and other examples)
- Handout and detailed “Cheat Sheet” available online
 - <http://www.cecs.uci.edu/~doemer/ESWeekTutorial.pdf>
 - <http://www.cecs.uci.edu/~doemer/ESWeekTutorial.txt>

Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

94

References (1)

- [Springer'20b] R. Dömer, Z. Cheng, D. Mendoza, E. Arasteh: "Pushing the Limits of Parallel Discrete Event Simulation for SystemC", in "A Journey of Embedded and Cyber-Physical Systems" by Jian-Jia Chen, Springer Nature, Switzerland, August 2020.
- [IJPP'20] Z. Cheng, T. Schmidt, R. Dömer: "Scaled Static Analysis and IP Reuse for Out-of-Order Parallel SystemC Simulation", International Journal of Parallel Programming (IJPP), Springer, June 2020.
- [DATE'20] D. Mendoza, Z. Cheng, E. Arasteh, R. Dömer: "Lazy Event Prediction using Defining Trees and Schedule Bypass for Out-of-Order PDES", Proceedings of DATE, Grenoble, France, March 2020.
- [ASPDAC'20] Z. Cheng, A. Arasteh, R. Dömer: "Event Delivery using Prediction for Faster Parallel SystemC Simulation", Proceedings of ASPDAC, Beijing, China, Jan. 2020.
- [Springer'20a] Z. Cheng, T. Schmidt, R. Dömer: "SystemC Coding Guideline for Faster Out-of-Order Parallel Discrete Event Simulation", chapter 6 in "Languages, Design Methods, and Tools for Electronic System Design" by T. Kazmierski, S. Steinhorst and D. Grosse, reprint of best papers at FDL 2018, Springer Nature, Switzerland, January 2020.
- [DVCon'19] D. Mendoza, A. Dingankar, Z. Cheng, R. Dömer: "Integrating Parallel SystemC Simulation into Simics® Virtual Platform", Proceedings of DVCon Europe, Munich, Germany, Oct. 2019.
- [TECS'19] Z. Cheng, R. Dömer: "Analyzing Variable Entanglement for Parallel Simulation of SystemC TLM-2.0 Models", ACM Transactions on Embedded Computing Systems (TECS), vol. 18, no. 5s, article 79, 20 pages, October 2019.

Tutorial at ESWEEK, Sept. 20, 2020

(c) 2020 R. Doemer et al., CECS

95

References (2)

- [CECS'19] G. Liu, T. Schmidt, Z. Cheng, D. Mendoza, R. Dömer: *"RISC Compiler and Simulator, Release V0.6.0: Out-of-Order Parallel Simulatable SystemC Subset"*, CECS TR 19-04, Sep. 2019.
- [IESS'19b] E. Arasteh, R. Dömer: *"An Untimed SystemC Model of GoogleLeNet"*, Proceedings of IESS, Springer, Friedrichshafen, Germany, Sep. 2019.
- [IESS'19a] Z. Cheng, T. Schmidt, R. Dömer: *"Enabling IP Reuse and Protection in Out-of-Order Parallel SystemC Simulation"*, Proceedings of IESS, Springer, Friedrichshafen, Germany, Sep. 2019.
- [FDL'18] Z. Cheng, T. Schmidt, R. Dömer: *"SystemC Coding Guideline for Faster Out-of-Order Parallel Discrete Event Simulation"*, Proceedings of FDL, Munich, Germany, Sep. 2018.
- [DATE'18] T. Schmidt, Z. Cheng, R. Dömer: *"Port Call Path Sensitive Conflict Analysis for Instance-Aware Parallel SystemC Simulation"*, Proceedings of DATE, Dresden, Germany, March 2018.
- [CECS'17] D. Mendoza, R. Dömer: *"A Tool for Visualization of SystemC Models"*, CECS Technical Report 17-06, Nov. 2017.
- [HLDVT'17] Z. Cheng, T. Schmidt, G. Liu, R. Dömer: *"Thread- and Data-Level Parallel Simulation in SystemC, a Bitcoin Miner Case Study"*, Proceedings of HLDVT, Santa Cruz, California, Oct. 2017.
- [DAC'17] T. Schmidt, G. Liu, R. Dömer: *"Towards Ultimate Parallel SystemC Simulation through Thread and Data Level Parallelism"*, Proceedings DAC, Austin, TX, June 2017.

References (3)

- [Springer'17] R. Dömer, G. Liu, T. Schmidt: *"Parallel Simulation"*, chapter 17 in *"Handbook of Hardware/Software Codesign"* by S. Ha and J. Teich, Springer Netherlands, June 2016.
- [ASPAC'17] T. Schmidt, G. Liu, R. Dömer: *"Hybrid Analysis of SystemC Models for Fast and Accurate Parallel Simulation"*, Proceedings ASPAC, Tokyo, Japan, January 2017.
- [IEEE ESL'16] R. Dömer: *"Seven Obstacles in the Way of Standard-Compliant Parallel SystemC Simulation"*, IEEE Embedded Systems Letters, vol. 8, no. 4, pp. 81-84, Dec. 2016.
- [DAC'15] R. Dömer: *"Towards Parallel Simulation of Multi-Domain System Models"*, Keynote, DAC workshop on System-to-Silicon Performance Modeling and Analysis, June 2015.
- [IEEE TCAD'14] W. Chen, X. Han, C. Chang, G. Liu, R. Dömer: *"Out-of-Order Parallel Discrete Event Simulation for Transaction Level Models"*, IEEE Transactions on CAD, vol. 33, no. 12, pp. 1859-1872, December 2014.
- [DATE'14] W. Chen, X. Han, R. Dömer: *"May-Happen-in-Parallel Analysis based on Segment Graphs for Safe ESL Models"*, Proceedings of DATE, Dresden, Germany, March 2014.
- [DATE'13] W. Chen, R. Dömer: *"Optimized Out-of-Order Parallel Discrete Event Simulation Using Predictions"*, Proceedings of DATE, Grenoble, France, March 2013.
- [DATE'12] W. Chen, X. Han, R. Dömer: *"Out-of-Order Parallel Simulation for ESL Design"*, Proceedings of DATE, Dresden, Germany, March 2012.