On the Limits of Standard-compliant Parallel Simulation of the IEEE SystemC Language

Forum on specification & Design Languages
Keynote

Rainer Dömer
doemer@uci.edu

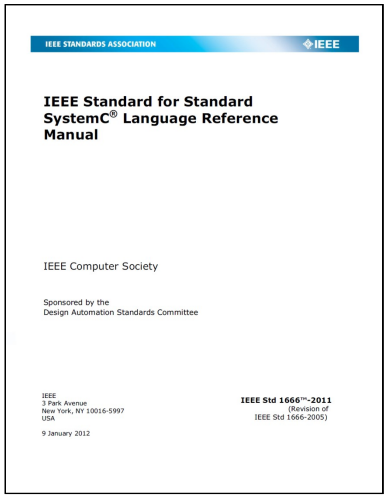
Center for Embedded and Cyber-Physical Systems
University of California, Irvine

UCIrvine
University of California, Irvine

IEEE Standard 1666-2011

- The SystemC Language
  - official standard
  - de-facto standard
- for
  - modeling
  - simulation
- of systems containing
  - hardware
  - software

➢ Keynote Focus
  - ➢ Parallelism in models
  - ➢ Parallelism in simulation
  - ➢ Standard compliance


IEEE STANDARDS ASSOCIATION IEEE

IEEE Standard for Standard SystemC® Language Reference Manual

IEEE Computer Society

Sponsored by the
Design Automation Standards Committee

IEEE
3 Park Avenue
New York, NY 10016-5997
USA

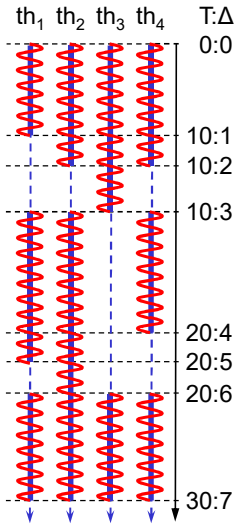
9 January 2012

IEEE Std 1666™-2011
(Revision of
IEEE Std 1666-2005)

FDL '18 Keynote, "Limits of Standard-compliant Parallel SystemC" (c) 2018 R. Doemer, CECS 2

Discrete Event Simulation (DES)

- SystemC uses 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
- Accellera Reference Simulator
  - Proof-of-concept implementation of IEEE 1666-2011 standard
  - ➢ A single thread is active at any time
    - ➢ Does not exploit parallelism
    - ➢ Cannot utilize multiple cores
  - ➢ Sequential simulation is slow



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]

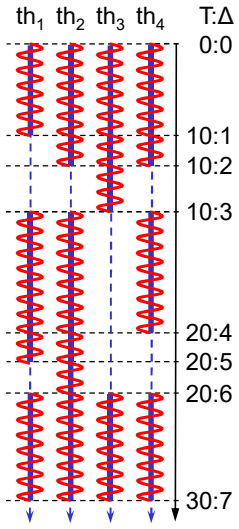
Sequential DE simulation is slow

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

Hardware-based Acceleration
- Sirowy et al. [DAC'10]
- Nanjundappa et al. [ASPDAC'10]
- Sinha et al. [ASPDAC'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
  - ➢ Order of magnitude speed up!
  - ➢ Problem solved!?
    - ➢ Not quite!
  - ➢ What about host platforms?
    - ➢ Multi- and many-core hosts are readily available
  - ➢ What about accuracy?
    - ➢ Is achievable with careful analysis
  - ➢ What about standard compliance?
    - ➢ That's where the problem is!



FDL '18 Keynote, "Limits of Standard-compliant Parallel SystemC" (c) 2018 R. Doemer, CECS 5

Problem Definition

- Given
  - Embedded systems are parallel
  - SystemC is suitable and popular for system design
  - Models exhibit explicit thread-level parallelism
  - Multi- and many-core host platforms are readily available
- Design
  - Fast Parallel Discrete Event Simulation
  - For the SystemC language
- Optimize
  - Maximize compliance with the IEEE 1666-2011 standard
- ➢ Why is this difficult?
  - ➢ 7 Obstacles stand in the way of standard-compliant parallel SystemC simulation [ESL'16]

FDL '18 Keynote, "Limits of Standard-compliant Parallel SystemC" (c) 2018 R. Doemer, CECS 6

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.

FDL '18 Keynote, "Limits of Standard-compliant Parallel SystemC" (c) 2018 R. Doemer, CECS 7

Parallel Discrete Event Simulation (PDES)

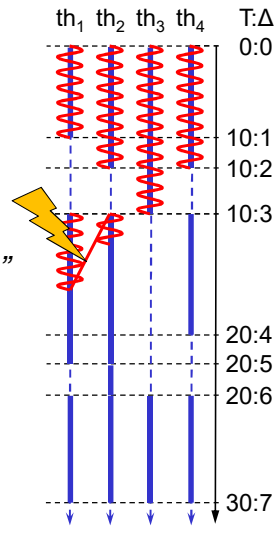
- Parallel DES [Fujimoto 1990]
  - Threads execute in parallel *iff*
    - in the same delta cycle, *and*
    - in the same time cycle
  - Order of magnitude speed up!

- 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)



FDL '18 Keynote, "Limits of Standard-compliant Parallel SystemC" (c) 2018 R. Doemer, CECS 8

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
  - Processes 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

FDL '18 Keynote, "Limits of Standard-compliant Parallel SystemC" (c) 2018 R. Doemer, CECS 9

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
  - *Entire* accessible simulator state needs attention…

FDL '18 Keynote, "Limits of Standard-compliant Parallel SystemC" (c) 2018 R. Doemer, CECS 10

Obstacle 2: Simulator State

- Fact: Discrete Event Simulation (DES) is presumed
- Problem: Parallel Discrete Event Simulation (PDES) is different from sequential DES
- Proposal: Carefully review simulator state primitives and revise as needed for PDES
  - *Entire* accessible simulator state needs attention
  - Special consideration for very strict semantics, e.g. debugging:
    Quote from IEEE 1666-2011, Section "4.2.1.2 Evaluation phase" (page 17):

    > The order in which process instances are selected from the set of runnable processes is implementation defined. However, if a specific version of a specific implementation runs a specific application using a specific input data set, the order of process execution shall not vary from run to run.

  - Sequential DES can remain valid as a special case of PDES
    - While PDES typically runs up to n threads in parallel, where n = number of cores on the host, we can set n = 1 to mimic the classic DES case

Obstacle 3: Lack of Thread Safety

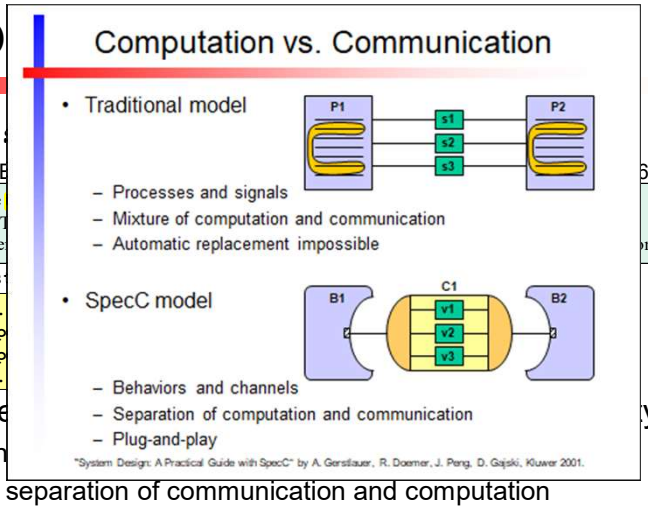
- Fact: Primitives are generally not multi-thread safe
  - Suspicious example from IEEE 1666-2011, page 194:

```
[...]
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

O

- Fact:
  - IEEE
  - sys
- Proble
  - Lan
    - 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)



FDL '18 Keynote, "Limits of Standard-compliant Parallel SystemC" (c) 2018 R. Doemer, CECS 13

Obstacle 4: Class sc_channel

- Proposal: Class **`sc_channel`**, derived from **`sc_module`**
  - Module encapsulates computation (hosts threads/processes)
  - Channel encapsulates communication (implemented interfaces)
  - Q: Why do we need channels? A: Thread safe communication!
  - Example: Blocking write in primitive channel **`sc_fifo.h`**

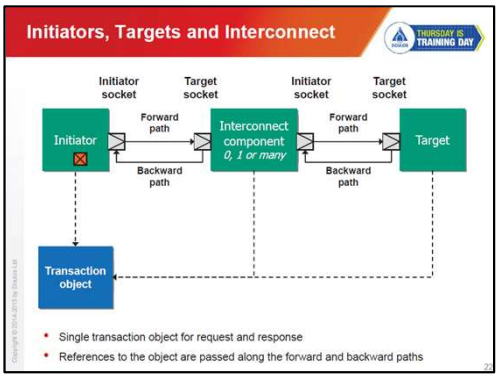
```
template <class T> inline
void sc_fifo<T>::write( const T& val_ )
{   sc_stacked_lock l(m_mutex);  // lock the channel mutex
    while( num_free() == 0 ) {
        sc_core::wait( m_data_read_event );
    }
    m_num_written ++;
    buf_write( val_ );
    request_update();
}
```

  - Race condition between **`num_free`** and **`m_num_written`**
  - Prevented by locking **`m_mutex`** of this channel instance
  - Channel acts as a *monitor* for multi-thread safe communication

FDL '18 Keynote, "Limits of Standard-compliant Parallel SystemC" (c) 2018 R. Doemer, CECS 14

Obstacle 5: TLM-2.0

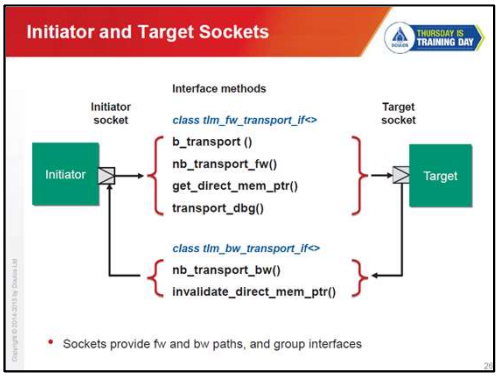
- Fact: Channel concept has disappeared
  - *"The Definitive Guide to SystemC: TLM-2.0 and the IEEE 1666-2011 Standard"*, Presentation by David Black, Doulos, at DAC'15 Training Day
- Problem: Where is the channel?



FDL '18 Keynote, "Limits of Standard-compliant Parallel SystemC" (c) 2018 R. Doemer, CECS 15

Obstacle 5: TLM-2.0

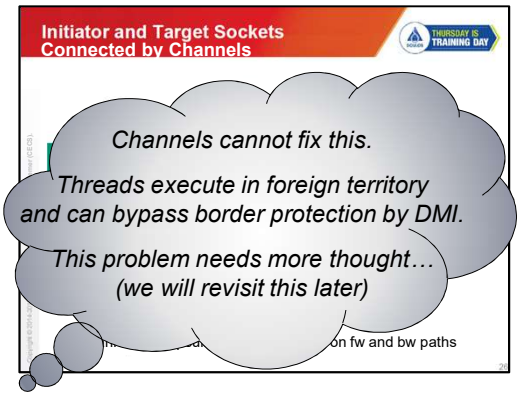
- Fact: Channel concept has disappeared
  - *"The Definitive Guide to SystemC: TLM-2.0 and the IEEE 1666-2011 Standard"*, Presentation by David Black, Doulos, at DAC'15 Training Day
- Problem: Where is the channel?
  - Interface methods are well-defined, but not contained
  - Separation of concerns *"Computation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



FDL '18 Keynote, "Limits of Standard-compliant Parallel SystemC" (c) 2018 R. Doemer, CECS 16

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
  - – Naïve Proposal: Encapsulate communication methods in channels



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, avoid **`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

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 [TCAD'14] is one solution…

Now what?

- *Seven Obstacles stand in the Way of Standard-Compliant Parallel SystemC Simulation*
  - Truly parallel and truly compliant SystemC appears elusive given the current IEEE standard

SystemC Evolution?

- *Seven Obstacles stand in the Way of Standard-Compliant Parallel SystemC Simulation*
  - SystemC Evolution Day 2016 [IEEE ESL'16]
- Let's overcome the identified 7 obstacles!
  - Move up from DES to PDES
  - Adopt a parallel mindset, expose and exploit parallelism
  - Apply the principle of separation of concerns
    - Modules encapsulate computation
    - Channels encapsulate communication
  - Simulate models faster with parallel execution semantics
- SystemC must evolve in a major revision (3.x)
  - C++11 already has built-in support for multithreading
  - SystemC must embrace true parallelism

Maximum Compliance with Standard

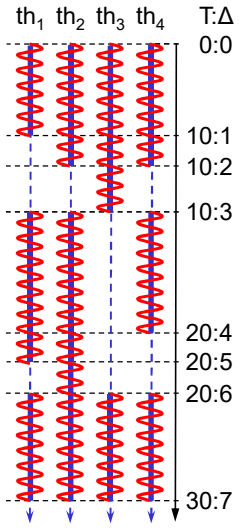
- *Seven Obstacles stand in the Way of Standard-Compliant Parallel SystemC Simulation*
  - SystemC Evolution Day 2016 [IEEE ESL'16]
- In absence of major changes to SystemC standard, let's make the best of it
  - Accept SystemC as it is (well, most of it)
  - Build the best parallel SystemC simulator possible
  - Aim for maximum compliance with the standard
- We took this risk, and created RISC!

Recoding Infrastructure for SystemC (RISC)

- 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
  - Thanks to Intel Corporation!

Recoding Infrastructure for SystemC (RISC)

- 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.
  - ➢ Significantly higher speedup!
    - Results at [DATE'12], [IEEE TCAD'14]
  - ➢ RISC compiler fully preserves…
    - ➢ Cause and effect relationship
    - ➢ Accuracy in results and timing



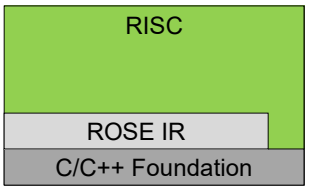
Recoding Infrastructure for SystemC (RISC)

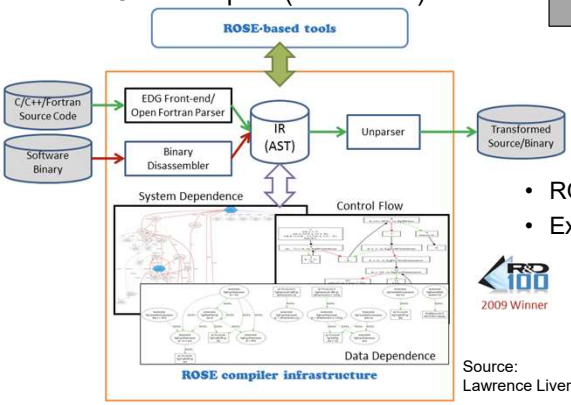
- Out-of-Order PDES Key Ideas
  1. Dedicated *SystemC compiler* with advanced model analysis
     - ➢ Static conflict analysis based on Segment Graphs
  2. *Parallel simulator* with out-of-order scheduling
     - ➢ Fast decision making at run-time, optimized mapping
- Fundamental Data Structure: *Segment 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

FDL '18 Keynote, "Limits of Standard-compliant Parallel SystemC" (c) 2018 R. Doemer, CECS 25

Dedicated SystemC Compiler

- 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

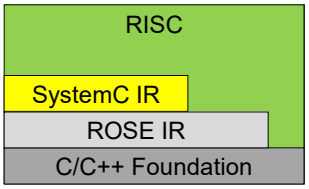
2009 Winner

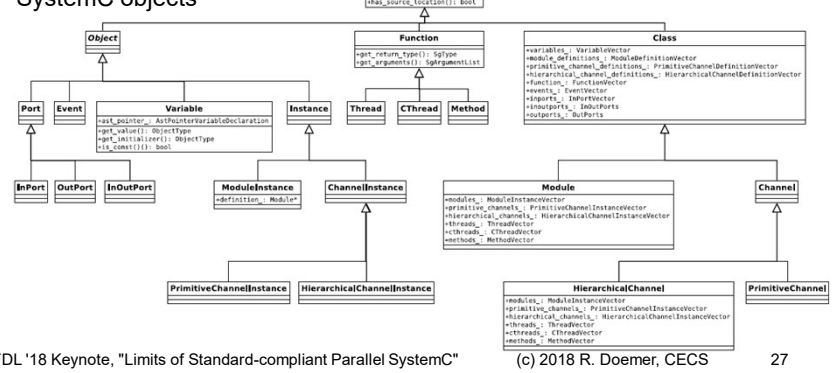
Source:
Lawrence Livermore National Laboratory (LLNL)

FDL '18 Keynote, "Limits of Standard-compliant Parallel SystemC" (c) 2018 R. Doemer, CECS 26

Dedicated SystemC Compiler

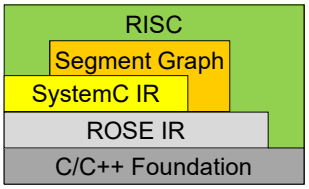
- RISC Software Stack
  - ➢ *Recoding Infrastructure for SystemC*
  - – SystemC Internal Representation
- Class hierarchy to represent SystemC objects

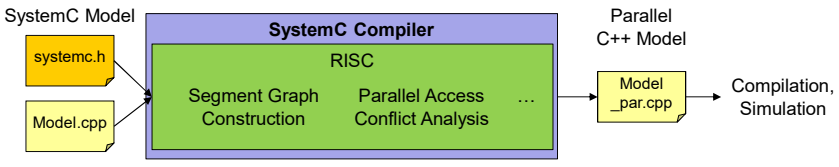


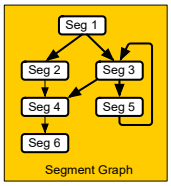


Dedicated SystemC Compiler

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



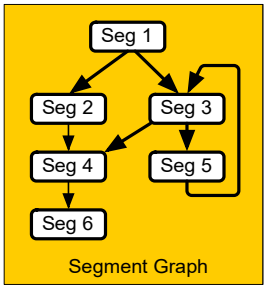




Step 1: Build a Segment Graph

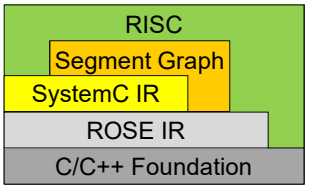
Dedicated SystemC Compiler

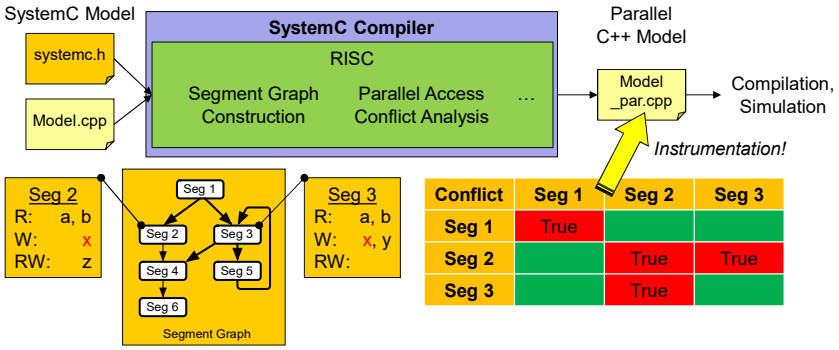
- Segment Graph
  - *Segment Graph* is a directed graph
    - Nodes: *Segments*
    - ➢ Code statements executed between two scheduling steps
      - Expression statements
      - Control flow statements (**`if`**, **`while`**, …)
      - Function calls
    - Edges: *Segment boundaries*
    - ➢ Primitives that trigger scheduler entry
      - **`wait(event)`**
      - **`wait(time)`**
  - ➢ Segment Graph is built automatically by the compiler [TCAD'14]
    - From the model source code
    - Via Abstract Syntax Tree and Control Flow Graph



Dedicated SystemC Compiler

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





Dedicated SystemC Compiler

- Segment Conflict Analysis
  - 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

Seg 2
R: a, b
W: x
RW: z

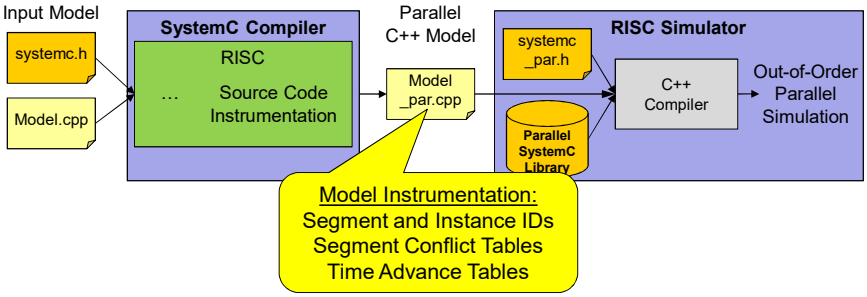
Seg 3
R: a, b
W: x, y
RW:

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

FDL '18 Keynote, "Limits of Standard-compliant Parallel SystemC" (c) 2018 R. Doemer, CECS 31

SystemC Compiler and Simulator

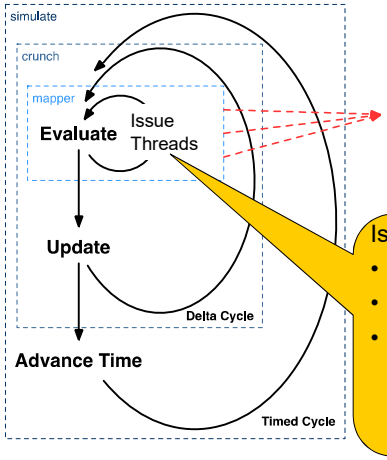
- Compiler and Simulator work hand in hand
  - Compiler performs conservative static analysis
  - Analysis results are passed to the simulator
  - Simulator can make safe scheduling decisions quickly
- ➢ Automatic Model Instrumentation
  - ➢ Static analysis results are inserted into the source code



FDL '18 Keynote, "Limits of Standard-compliant Parallel SystemC" (c) 2018 R. Doemer, CECS 32

Parallel SystemC Simulator

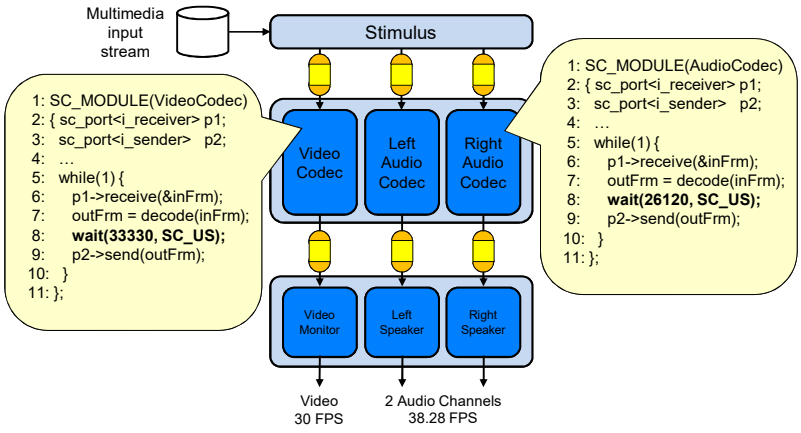
- Simulator kernel with Out-of-Order Parallel Scheduler
  - Conceptual OoO PDES execution



Issue threads…
- truly in *parallel* and *out-of-order*
- whenever they are *ready*
- and have *no conflicts*!
  - Fast conflict table lookup
  - Optimized thread-to-core mapping

Experiments and Results

- DVD Player Example
  - Parallel video and audio decoding with different frame rates

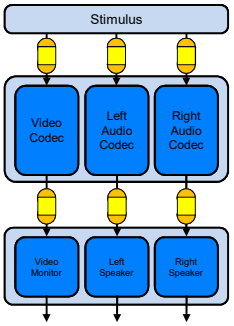



```
1: SC_MODULE(VideoCodec)
2: { sc_port<i_receiver> p1;
3:   sc_port<i_sender>   p2;
4:   …
5:   while(1) {
6:     p1->receive(&inFrm);
7:     outFrm = decode(inFrm);
8:     wait(33330, SC_US);
9:     p2->send(outFrm);
10:  }
11: };
```

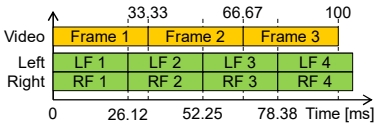
```
1: SC_MODULE(AudioCodec)
2: { sc_port<i_receiver> p1;
3:   sc_port<i_sender>   p2;
4:   …
5:   while(1) {
6:     p1->receive(&inFrm);
7:     outFrm = decode(inFrm);
8:     wait(26120, SC_US);
9:     p2->send(outFrm);
10:  }
11: };
```

Experiments and Results

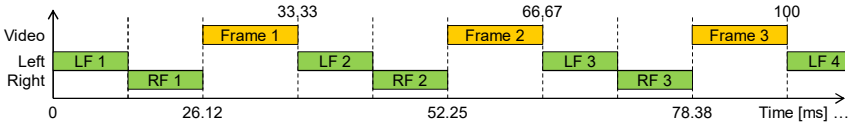
- DVD Player Example
  - Parallel video and audio decoding with different frame rates



1. Real time schedule: fully parallel

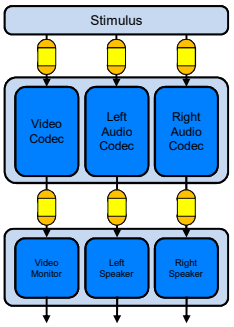


2. Reference simulator schedule (DES)

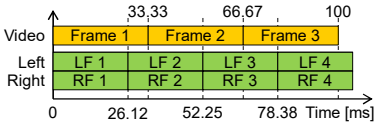


Experiments and Results

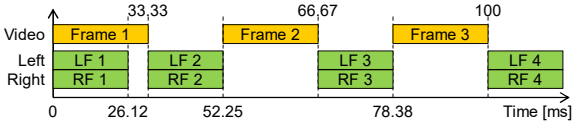
- DVD Player Example
  - Parallel video and audio decoding with different frame rates



1. Real time schedule: fully parallel



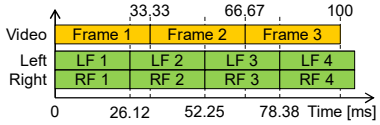
3. Synchronous parallel schedule (PDES)



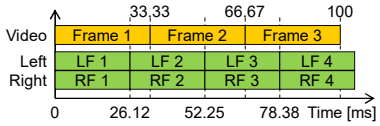
Experiments and Results

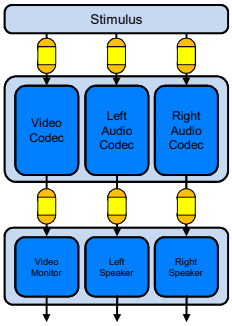
- DVD Player Example
  - Parallel video and audio decoding with different frame rates

1. Real time schedule: fully parallel



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

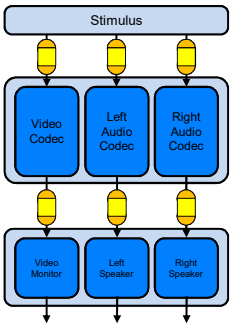




Experiments and Results

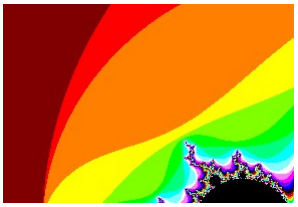
- DVD Player Example
  - Parallel video and audio decoding with different frame rates
- Simulator Run Times
  - 4-core Intel® Xeon® CPU at 3.4 GHz
  - RISC v0.2.1, Posix-threads

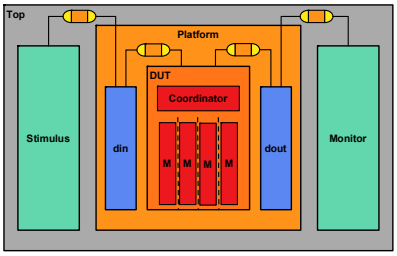
		DES	PDES	OoO PDES
10 sec stream	Run Time	6.98 s	4.67 s	2.94 s
	CPU Load	97%	145%	238%
	Speedup	1 x	1.49 x	2.37 x
100 sec stream	Run Time	68.21 s	45.91 s	28.13 s
	CPU Load	100%	149%	251%
	Speedup	1 x	1.49 x	2.42 x



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
    - Embarrassingly parallel
      - Parallelism at pixel level
  - SystemC Model
    - TLM abstraction
    - Horizontal image slices
    - Highly configurable
    - Parallelism parameter from 1 to 256 slices

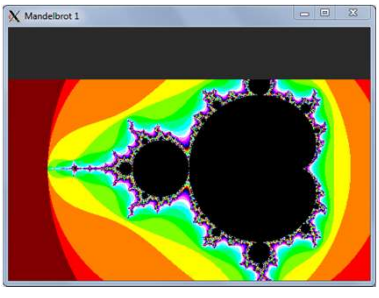


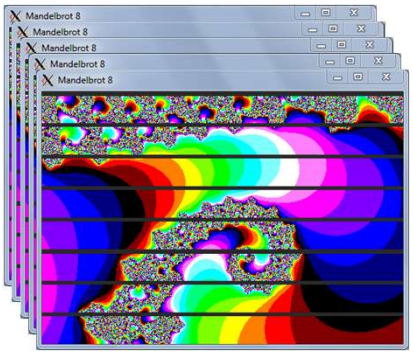


FDL '18 Keynote, "Limits of Standard-compliant Parallel SystemC" (c) 2018 R. Doemer, CECS 39

Experiments and Results

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





FDL '18 Keynote, "Limits of Standard-compliant Parallel SystemC" (c) 2018 R. Doemer, CECS 40

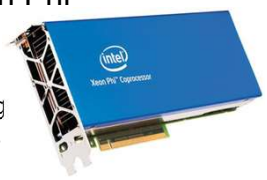
Experiments and Results

- 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

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

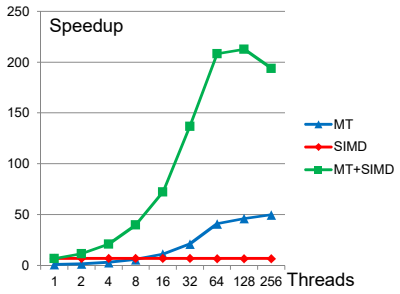


- ➢ RISC extended for exploiting 2 types of parallelism
  - Out-of-Order PDES: thread-level parallelism
  - Intel® compiler SIMD: data-level parallelism
  - ➢ RISC SIMD Advisor identifies functions with data-level parallelism suitable for SIMD vectorization
  - ➢ DAC '17 paper: *"Exploiting Thread and Data Level Parallelism for Ultimate Parallel SystemC Simulation"*

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!**

RISC Open Source Software

- RISC Compiler and Simulator are freely available
  - http://www.cecs.uci.edu/~doemer/risc.html#RISC042
    - Installation notes and script: **`INSTALL`**, **`Makefile`**
    - Open source tar ball: **`risc_v0.4.2.tar.gz`**
    - Docker script and container: **`Dockerfile`**
    - Doxygen documentation: RISC API, OOPSC API
    - Tool manual pages: **`risc`**, **`simd`**, **`visual`**, …
    - BSD license terms: **`LICENSE`**
  - Companion Technical Report
    - CECS Technical Report 17-05: **`CECS_TR_17_05.pdf`**

➢ Docker container:

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

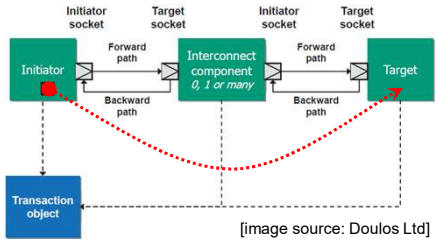
  - ➢ https://hub.docker.com/r/ucirvinelecs/risc/

Ongoing Efforts: Scaling RISC

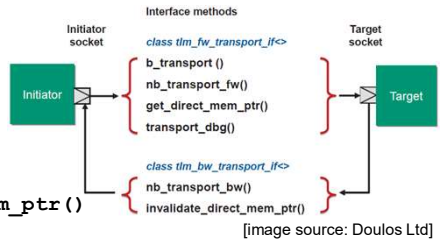
- Support for Industrial Sizes and Design Flows
  1. New concept of Partial Segment Graphs (PSG)
     - File hierarchies with multiple translation units
     - Support for 3rd party libraries, IP protection
  2. Improved compiler analysis for less false conflicts
     - Port-Call-Path technique identifies instances [DATE'18]
     - Reference type analysis identifies target variables
  3. Evaluation of RISC in industry
     - "Big example", very large SystemC model at RTL abstraction
     - Integration with Simics virtual platforms
  4. Support for TLM-2.0
     - Pro: Part of SystemC standard, needed for wide RISC adoption
     - Con: Obstacle 5!
       No channel, unprotected execution in foreign territory

Scaling RISC: Overcoming Obstacle 5

- SystemC TLM-2.0
  - Initiators and Targets
    - Sockets
    - Forward path
    - Backward path
    - Shared transaction object
    - DMI bypass
  - Well-defined Socket API
    1. `b_transport()`
    2. `nb_transport_fw()`
    3. `nb_transport_bw()`
    4. `transport_dbg()`
    5. `get_direct_mem_ptr()`
    6. `invalidate_direct_mem_ptr()`



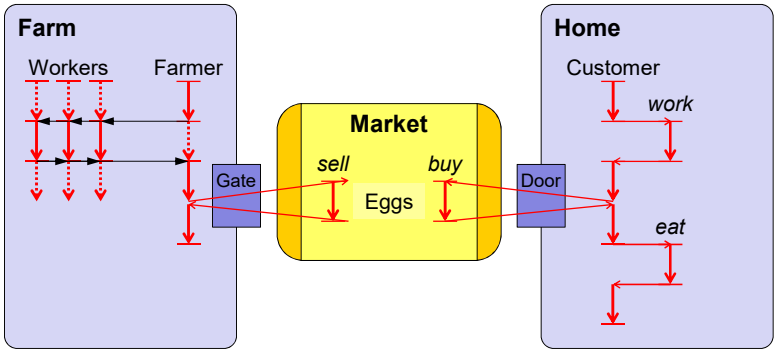
[image source: Doulos Ltd]



[image source: Doulos Ltd]

Scaling RISC: Overcoming Obstacle 5

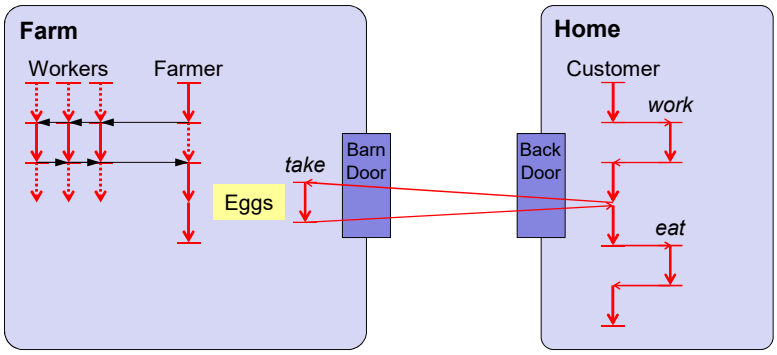
- Classic TLM: Producer-Consumer Example



  - Threads operate in their own modules or protected channels
    - Well-behaved execution in safe execution contexts
    - Current RISC analysis fully supports this modeling style

Scaling RISC: Overcoming Obstacle 5

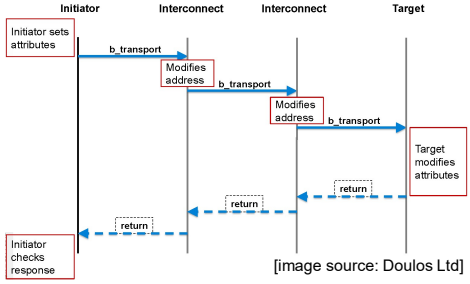
- New TLM-2.0: Producer-Consumer Example



  - No channels! Threads operate directly in others' modules
    - Fast, but dangerous execution in foreign territory
    - Current RISC analysis cannot handle this modeling style

Scaling RISC: Overcoming Obstacle 5

- TLM-2.0 is quite different from traditional TLM
  - ➢ Need significant extension of compiler analysis
    - Conflict analysis must follow the initiator's threads execution through the hierarchical model structure to the targets



[image source: Doulos Ltd]

  - ➢ Leverage and extend existing RISC technology
    - Instance tree, instance path, and instance ID [RISCv020]
    - Hybrid analysis [ASPDAC'17]
    - Port-Call-Paths (PCP) [DATE'18]

Concluding Remarks

- On the Limits of Standard-compliant Parallel Simulation of SystemC
  - – Seven Obstacles stand in the way of parallel SystemC
    - Co-routine semantics, sequential simulator state primitives, lack of thread-safety, weak role of channels, TLM-2.0, temporal decoupling, and an overall sequential modeling mindset
  - ➢ Truly parallel SystemC appears elusive given the current IEEE Standard 1666-2011
- Parallel Simulation with Maximum Compliance
  - – Example: Recoding Infrastructure for SystemC (RISC)
    - Out-of-order Parallel Discrete Event Simulation
    - Dedicated SystemC compiler and parallel simulator
    - Multi- and many-core host platforms
    - Two orders of magnitude faster simulation with full accuracy
    - Open source

Acknowledgments

- For solid work, fruitful discussions, and honest feedback, I would like to thank:
  - My team at UCI
    - Zhongqi Cheng
    - Guantao Liu
    - Daniel Mendoza
    - Tim Schmidt
  - 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!

Selected References

- [RISC v0.4.2] RISC project, release 0.4.2: http://www.cecs.uci.edu/~doemer/risc.html
- [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] G. Liu, T. Schmidt, Z. Cheng, R. Dömer: *"RISC Compiler and Simulator, Release V0.4.0: Out-of-Order Parallel Simulatable SystemC Subset"*, CECS TR 17-05, July 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.
- [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.
- [ASPDAC'17] T. Schmidt, G. Liu, R. Dömer: *"Hybrid Analysis of SystemC Models for Fast and Accurate Parallel Simulation"*, Proceedings ASPDAC, 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'12] W. Chen, X. Han, R. Dömer: *"Out-of-Order Parallel Simulation for ESL Design"*, Proceedings of DATE, Dresden, Germany, March 2012.