Attacking the System Validation Challenge with Advanced Parallel Simulation: The Good News and the Bad News

Keynote at HLDVT, Oct. 5, 2017

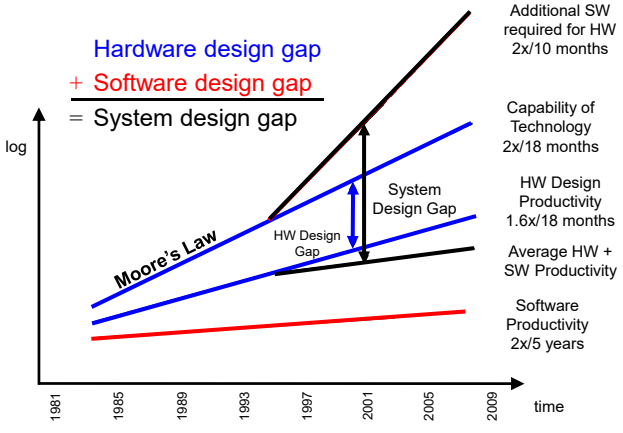
Rainer Dömer
doemer@uci.edu

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

UCIrvine
University of California, Irvine

System Validation Challenge

- Electronic System Level (ESL) Complexity
  - is rising exponentially



(source: *"Hardware-dependent Software"*, Ecker et al., 2009)

System Validation Challenge

- Electronic System Level (ESL) Complexity
  - is rising exponentially
- Validation Challenge
  - ➢ Efficient simulation must be fast and accurate!
- Facts
  - Parallel Discrete Event Simulation (PDES)
    - Known for several decades [Fujimoto1990]
    - Multi- and many-core host platforms are readily available
  - ESL Design Models
    - Models explicitly exhibit thread-level parallelism
    - SystemC standard is established [IEEE 1666-2011]
- Question:
  - ➢ Does Advanced Parallel Simulation meet the challenge?

System Validation Challenge

- Attacking the Validation Challenge with
  Advanced Parallel Simulation of SystemC Models

➢ Good News
- SystemC is a common base language for modeling
- Many approaches exist to leverage parallel processing
- Example:
  - Out-of-Order Parallel Simulation with RISC
  - Several orders of magnitude speedup (200x)

➢ Bad News
- IEEE SystemC standard makes it difficult
  to exploit parallelism
- 7 Obstacles stand in the way
  of standard-compliant parallel SystemC simulation

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]

Discrete Event 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]
- ➢ *OoO PDES with RISC*

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

HLDVT '17 Keynote, "Attacking the System Validation Challenge" (c) 2017 R. Doemer, CECS 5

Recoding Infrastructure for SystemC (RISC)

- Advanced Parallel SystemC Simulation
  - Out-of-Order PDES on many-core host platforms
  - Maximum compliance with IEEE SystemC semantics
- Introduction of a Dedicated SystemC Compiler
  - Recoding Infrastructure for SystemC (RISC)
  - Advanced conflict analysis for safe parallel execution
  - Model instrumentation and code generation
- Parallel SystemC Kernel
  - Out-of-order parallel scheduler, multi-thread safe primitives
  - Multi- and many-core target platform (e.g. Intel® Xeon Phi™)
- Open Source
  - Collaboration with Accellera SystemC Language WG
  - Result of a project sponsored by Intel Corp.

HLDVT '17 Keynote, "Attacking the System Validation Challenge" (c) 2017 R. Doemer, CECS 6

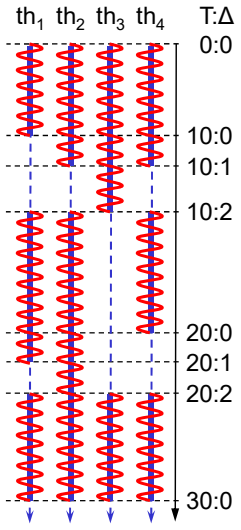
Out-of-Order PDES Technology

- ➢ *SystemC Simulation must be Fast and Accurate!*
- Traditional Discrete Event Simulation (DES)
  - Reference simulators run *sequentially*, only one thread at a time (cooperative multi-threading model)
  - Cannot utilize the capabilities of multi- or many-core hosts
- Parallel Discrete Event Simulation (PDES)
  - Threads run in *parallel* (if at the same delta cycle and time)
  - Simulation-cycles are absolute barriers
- Out-of-order Parallel DE Simulation (OoO PDES)
  - Threads run in *parallel and out-of-order* [DATE'12, TCAD'14] even in different delta and time cycles if there are no conflicts
  - Aggressive, runs maximum number of threads in parallel, but *fully preserves DES semantics and model accuracy*

HLDVT '17 Keynote, "Attacking the System Validation Challenge" (c) 2017 R. Doemer, CECS 7

Discrete Event Simulation (DES)

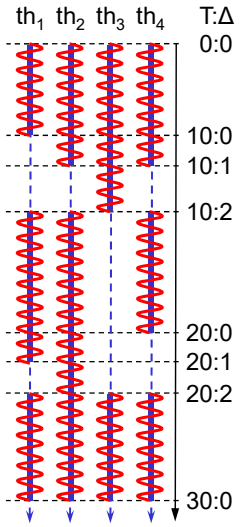
- Traditional DES
  - Concurrent threads of execution
  - Managed by a central scheduler
  - Driven by events and time advances
    - Delta-cycle
    - Time-cycle
  - ➢ Partial temporal order with barriers
- Sequential Reference Simulator
  - IEEE SystemC standard states cooperative multi-threading
  - ➢ A single thread is active at any time!
  - ➢ Cannot exploit parallelism
  - ➢ Cannot utilize multiple cores



HLDVT '17 Keynote, "Attacking the System Validation Challenge" (c) 2017 R. Doemer, CECS 8

Parallel Discrete Event Simulation (PDES)

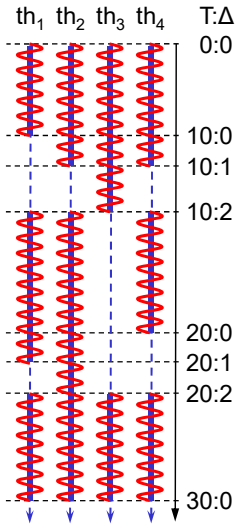
- Parallel DES
  - Threads execute in parallel *iff*
    - in the same delta cycle, *and*
    - in the same time cycle
  - ➢ Significant speed up!
  - *Synchronous* PDES: Cycle boundaries are *absolute barriers*!
- Aggressive Parallel DES
  - Conservative Approaches
    - Careful static analysis prevents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



HLDVT '17 Keynote, "Attacking the System Validation Challenge" (c) 2017 R. Doemer, CECS 9

Out-of-Order PDES Technology

- Out-of-Order Parallel DES
  - ➢ Breaks synchronization barrier!
  - Threads execute in parallel *iff*
    - in the same delta cycle, *and*
    - in the same time cycle,
    - *OR* if there are no conflicts!
  - ➢ Allows as many threads in parallel as possible
  - ➢ Significantly higher speedup!
    - Results at [DATE'12], [IEEE TCAD'14]
  - ➢ Advanced compiler fully preserves…
    - ➢ DES execution semantics
    - ➢ Accuracy in results and timing



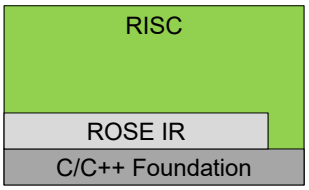
HLDVT '17 Keynote, "Attacking the System Validation Challenge" (c) 2017 R. Doemer, CECS 10

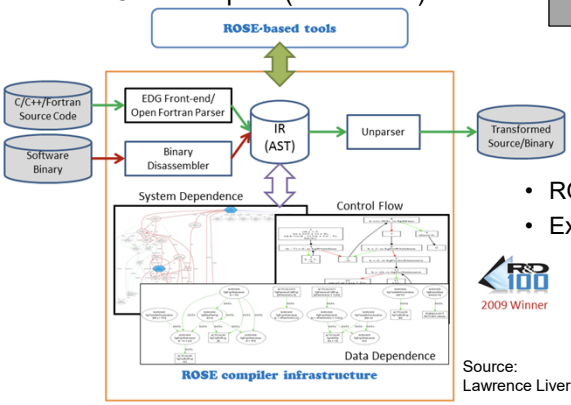
Out-of-Order PDES Technology

- OoO 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 on many cores
     - ➢ 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**)
  - – Journal publication: *"OoO PDES for TLM"* [IEEE TCAD'14]
    - Comprehensive article with HybridThreads extension

Dedicated SystemC Compiler

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





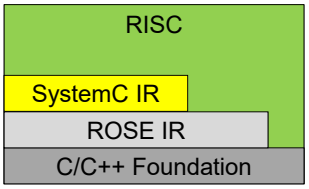
- ROSE Internal Representation
- Explicit support for
  - Source code analysis
  - Source-to-source transformations

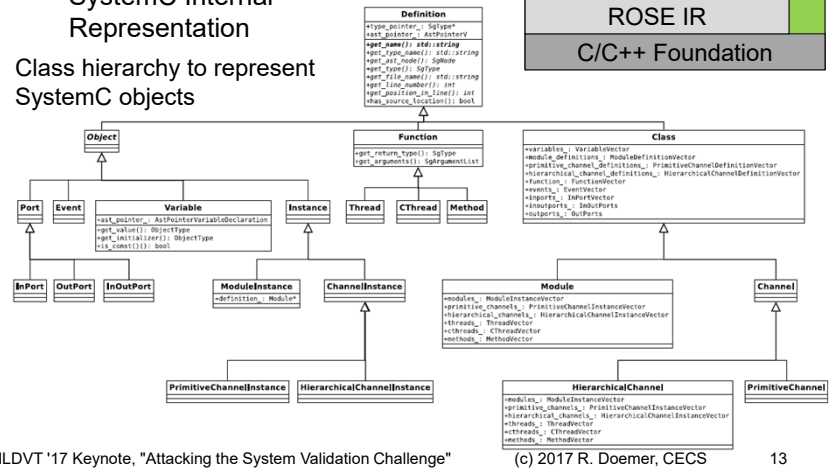
2009 Winner

Source:
Lawrence Livermore National Laboratory (LLNL)

Dedicated SystemC Compiler

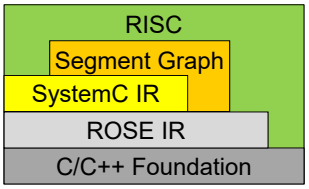
- RISC Software Stack
  - ➢ *Recoding Infrastructure for SystemC*
  - – SystemC Internal Representation
- Class hierarchy to represent SystemC objects

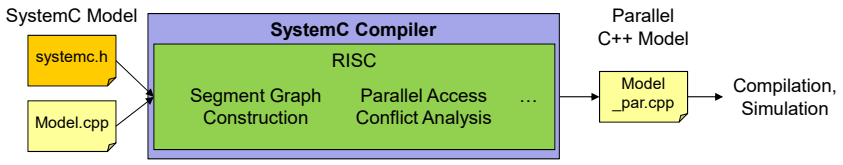


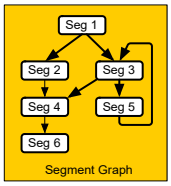


Dedicated SystemC Compiler

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



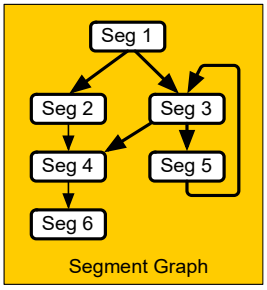




Step 1: Build a Segment Graph

Dedicated SystemC Compiler

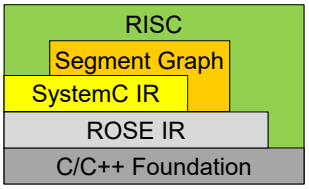
- Segment Graph
  - *Segment Graph* is a directed graph
    - Nodes: *Segments*
    - ➢ Code statements executed between two scheduling steps
      - Expression statements
      - Control flow statements (**`if`**, **`while`**, …)
      - Function calls
    - Edges: *Segment boundaries*
    - ➢ Primitives that trigger scheduler entry
      - **`wait(event)`**
      - **`wait(time)`**

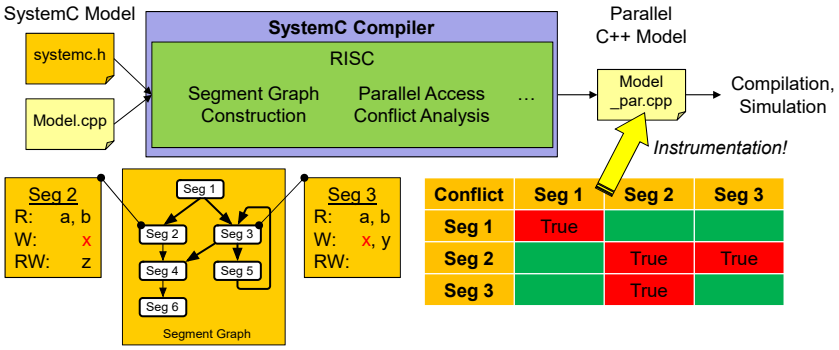


  - ➢ Segment Graph is built automatically by the compiler [TCAD'14]
    - From the model source code
    - Via the Abstract Syntax Tree and Control Flow Graph

Dedicated SystemC Compiler

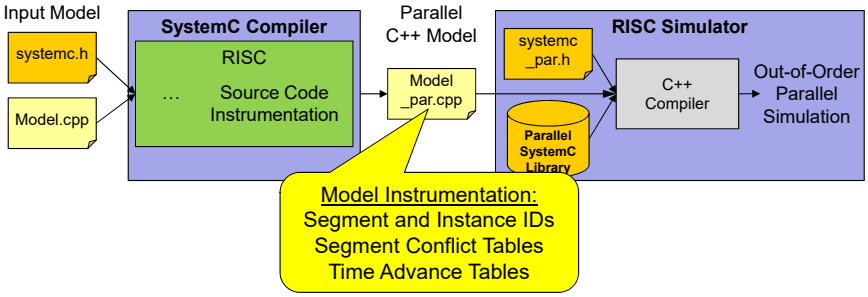
- RISC Software Stack
  - ➢ *Recoding Infrastructure for SystemC*
  - 1) Segment Graph construction
  - 2) Parallel access conflict analysis
  - 3) Model instrumentation





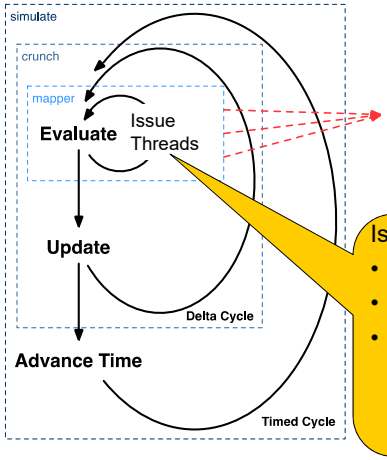
SystemC Compiler and Simulator

- 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



Parallel SystemC Simulator

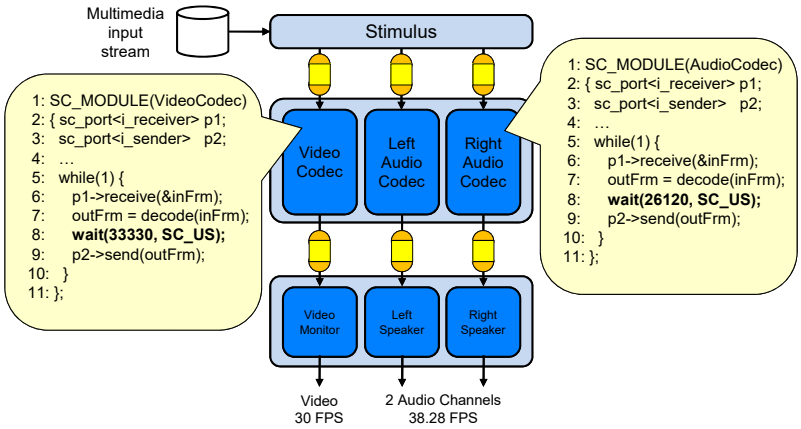
- Simulator kernel with Out-of-Order Parallel Scheduler
  - Conceptual OoO PDES execution



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

Experiments and Results

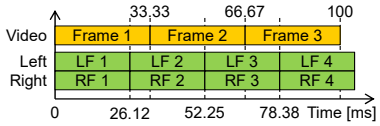
- DVD Player Example
  - Parallel video and audio decoding with different frame rates

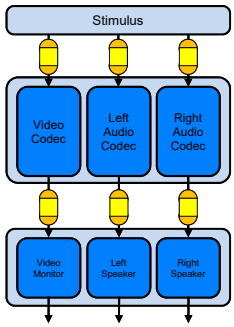


Experiments and Results

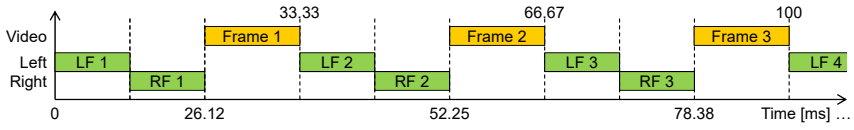
- DVD Player Example
  - Parallel video and audio decoding with different frame rates

1. Real time schedule: fully parallel



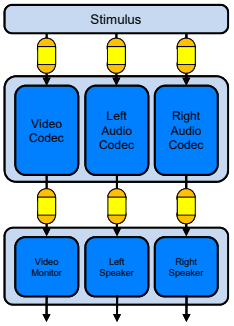


2. Reference simulator schedule (DES)

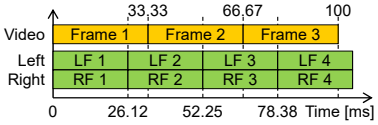


Experiments and Results

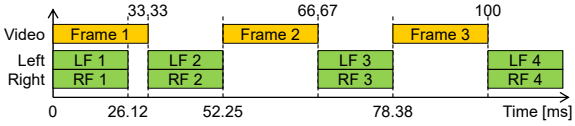
- DVD Player Example
  - Parallel video and audio decoding with different frame rates



1. Real time schedule: fully parallel

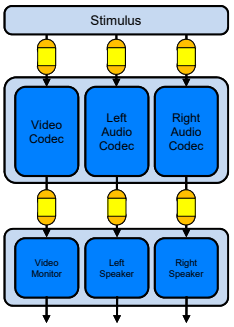


3. Synchronous parallel schedule (PDES)

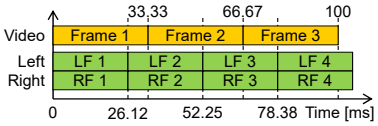


Experiments and Results

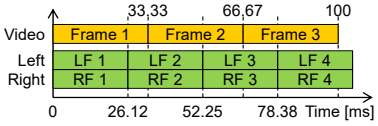
- DVD Player Example
  - Parallel video and audio decoding with different frame rates



1. Real time schedule: fully parallel

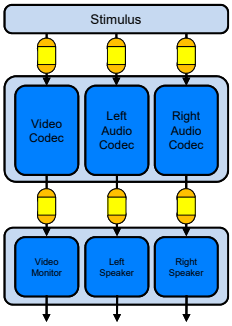


4. Out-of-order parallel schedule (OoO PDES)



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

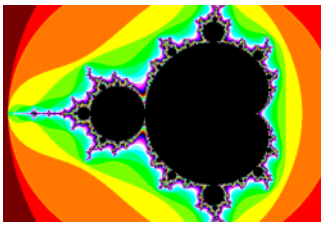


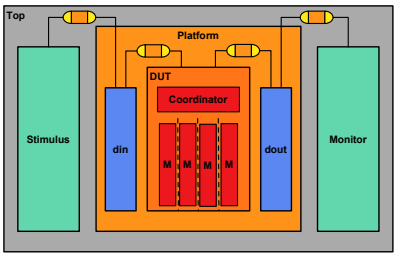
		DES	PDES	OoO PDES
10 sec stream	Run Time	6.98 s	4.67 s	2.94 s
	CPU Load	97%	145%	238%
	Speedup	1 x	1.49 x	2.37 x
100 sec stream	Run Time	68.21 s	45.91 s	28.13 s
	CPU Load	100%	149%	251%
	Speedup	1 x	1.49 x	2.42 x

HLDVT '17 Keynote, "Attacking the System Validation Challenge" (c) 2017 R. Doemer, CECS 23

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





HLDVT '17 Keynote, "Attacking the System Validation Challenge" (c) 2017 R. Doemer, CECS 24

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

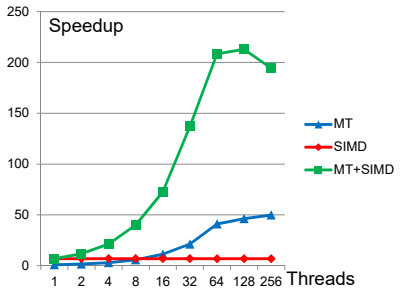
- Many-Core Target Platform: Intel® Xeon Phi™
  - Many Integrated Core (MIC) architecture
    - 1 Coprocessor 5110P CPU at 1.052 GHz
    - 60 physical cores with 4-way hyper-threading
      - Appears as regular Linux host with 240 cores
    - Up to 8 lanes available for vector processing

- ➢ RISC extended for exploiting *two* 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
  - 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

- Out-of-Order Parallel SystemC Compiler and Simulator
  - Open source release V0.4.0 (July 31, 2017)
    - Installation notes: **`INSTALL`**
    - Installation Makefile: **`Makefile`**
    - Open source tar ball: **`risc_v0.4.0.tar.gz`**
    - Doxygen documentation: RISC API
    - Doxygen documentation: OoO Parallel SystemC API
    - Tool manual pages: **`risc`**, **`elab`**, **`simd`**
    - BSD license terms: **`LICENSE`**
  - Companion Technical Report released July 31, 2017
    - CECS Technical Report 17-05: **`CECS_TR_17_05.pdf`**

➢ Available for download now. Give it a try!

➢ http://www.cecs.uci.edu/~doemer/risc.html#RISC040

Summary of the Good News

- Recoding Infrastructure for SystemC (RISC)
  - Advanced parallel SystemC simulation and modeling
  - Out-of-order PDES on multi- and many-core host platforms
  - Dedicated SystemC compiler and parallel simulation library
  - Automatic conflict analysis based on Segment Graphs (SG)
  - *Maximum* compliance with IEEE SystemC semantics
  - Open source freely available, v0.4.0 released July 2017
- Future Work
  - Scaling to multiple translation units (partial Segment Graphs)
  - Virtual Platform (VP) integration (e.g. Simics VP)
  - TLM-2.0 support
    - ➢ *And that's where the Bad News starts…*

The Bad News

- ➢ *Seven Obstacles stand in the Way of Standard-Compliant Parallel SystemC Simulation*
  - Presentation at SystemC Evolution Day 2016 [IEEE ESL'16]
    - Accellera meeting on the next generation of IEEE SystemC
- Truly parallel simulation of SystemC models violates the IEEE 1666-2011 standard
  - Parallel execution is *not* standard compliant!
- Careful technical review and evaluation of
  - Standard SystemC® Language Reference Manual
    - IEEE Std 1666™-2011 (Revision of IEEE Std 1666-2005)
  - Accellera open source proof-of-concept library (v2.3.1)

…has identified 7 obstacles and potential solutions

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.

HLDVT '17 Keynote, "Attacking the System Validation Challenge" (c) 2017 R. Doemer, CECS 31

Parallel Discrete Event Simulation (PDES)

- SystemC LRM Requirement:
  "The scheduler is not pre-emptive."

```
int x;  // global variable

void thread1()      void thread42()
{ x = 0;            { x = 7;
  x = x + 1;          x = x * 6;
  cout << x;          cout << x;
}                   }
```

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

HLDVT '17 Keynote, "Attacking the System Validation Challenge" (c) 2017 R. Doemer, CECS 32

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

HLDVT '17 Keynote, "Attacking the System Validation Challenge" (c) 2017 R. Doemer, CECS 33

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…

HLDVT '17 Keynote, "Attacking the System Validation Challenge" (c) 2017 R. Doemer, CECS 34

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
  - The general notion of *shared 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.

  - Strict 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

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.

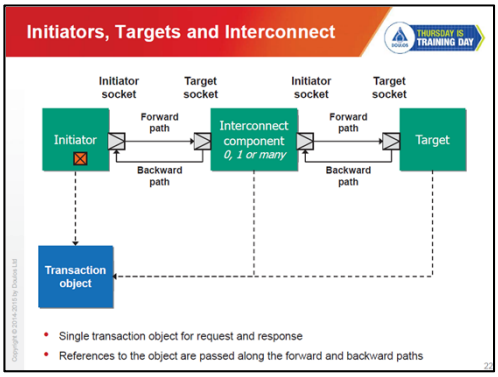
  - **`systemc-2.3.1/include/sysc/kernel/sc_module.h`**

```
[...]
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)

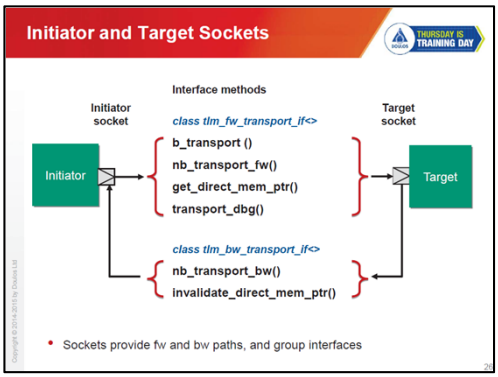
Obstacle 5: TLM-2.0

- 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?



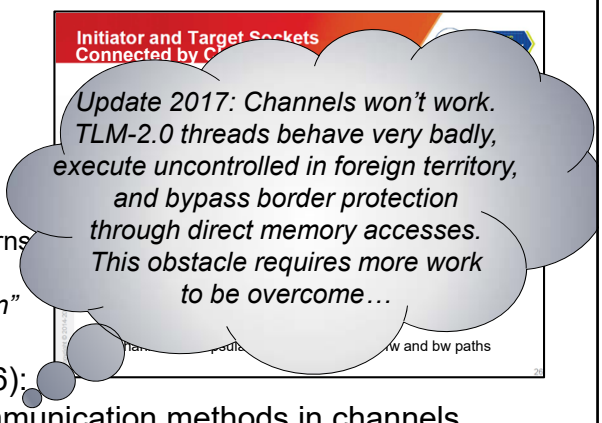
Obstacle 5: TLM-2.0

- 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



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 (in 2016): 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, 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

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)

Concluding the Bad News

- Parallel SystemC violates the current IEEE standard
  - Parallel simulation cannot be IEEE 1666-2011 compliant
- We must 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
    - *Otherwise it will go down the same path as the dinosaurs…*

System Validation Challenge

- Attacking the Validation Challenge with
  Advanced Parallel Simulation of SystemC Models
- Good News
  - SystemC is a common base language for modeling
  - Many approaches exist to leverage parallel processing
  - Example:
    - Out-of-Order Parallel Simulation with RISC
    - Several orders of magnitude speedup (200x)
- Bad News
  - IEEE SystemC standard makes it difficult
    to exploit parallelism
  - 7 Obstacles stand in the way
    of standard-compliant parallel SystemC simulation

Acknowledgments

- For solid work, fruitful discussions, and honest feedback, I would like to thank:
  - My team at UCI
    - Guantao Liu
    - Tim Schmidt
    - Zhongqi Cheng
  - Our collaborators at Intel
    - Ajit Dingankar
    - Desmond Kirkpatrick
    - Abhijit Davare
    - Philipp Hartmann

This work has been supported in part by substantial funding from Intel Corporation.

Selected References

- [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.
- [HLDVT'16] G. Liu, T. Schmidt, R. Dömer: *"A Segment-Aware Multi-Core Scheduler for SystemC PDES"*, Proceedings of HLDVT, Santa Cruz, California, October 2016.
- [Accellera'16] R. Dömer: *"Seven Obstacles in the Way of Parallel SystemC Simulation"*, Presentation at SystemC Evolution Day 2016, Munich, Germany, May 2016.
- [ASPDAC'15] G. Liu, T. Schmidt, R. Dömer, A. Dingankar, D. Kirkpatrick: *"Optimizing Thread-to-Core Mapping on Manycore Platforms with Distributed Tag Directories"*, Proceedings of ASPDAC, Tokyo, Japan, January 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.