RISC: A Compiler for Parallel SystemC with Maximum Standard Compliance

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

1

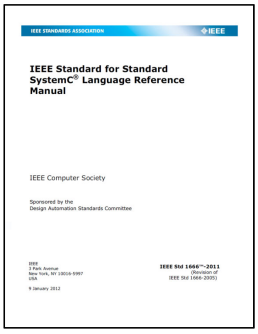
Presentation Copyright Permission

– A non-exclusive, irrevocable, royalty-free copyright permission is granted by **Rainer Doemer, CECS,** to use this material in developing all future revisions and editions of the resulting draft and approved Accellera Systems Initiative **SystemC** standard, and in derivative works based on this standard.

2

Standard-Compliant Parallel SystemC

- 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)

3

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

Alternative: SystemC Compiler!
Use static analysis to prevent parallel access conflicts.

4

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

Yes: Carefully review and revise the SystemC API!
Adjust the proof-of-concept simulator accordingly.

5

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

Yes: Multi-thread safe SystemC primitives!
Adjust the proof-of-concept simulator accordingly.

6

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;
[...]
```

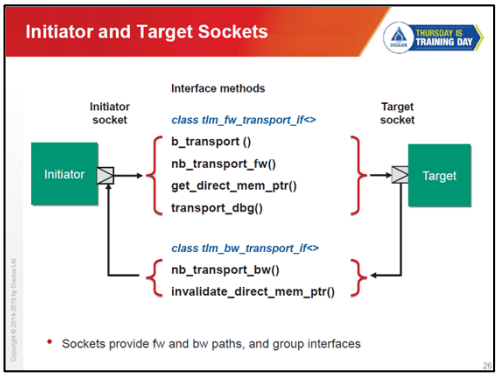
- Problem: Alias type is only another name, no new type
  - Language does not distinguish modules and channels
  - No separation of communication and computation
    - Breaks a key system-level design principle…
- Proposal: Class **`sc_channel`**, derived from **`sc_module`**

Yes: Derive sc_channel from sc_module. Fixed!

7

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:



No: Compiler can ensure safety in TLM-2.0!

8

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

No: SC_METHOD is fine!
Compiler can recode them to true parallel threads.
Yes: Parallel mindset! SystemC evolution!

9

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)`**

Not sure… Future Work!
Compiler analysis can likely help here as well.

10

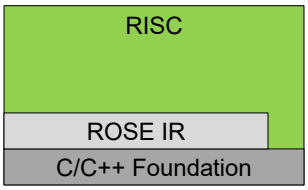
A Compiler-Based Approach

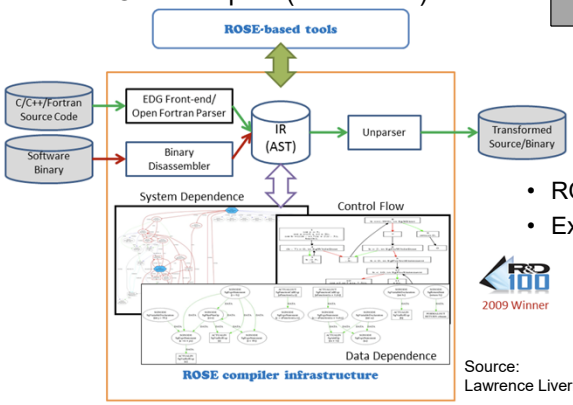
- While the SystemC standard has not changed, my group has worked hard
  - ➢ *"Let's make the best of it!"*
- Goals
  - Accept SystemC as it is (well, most of it)
  - Build the best parallel SystemC simulator possible
  - *Aim for maximum compliance with the standard*
- ➢ We took this risk, and created RISC!
  - ➢ *Recoding Infrastructure for SystemC*
- ➢ A dedicated **SystemC compiler** with parallel **SystemC simulator** can overcome the 7 obstacles …

11

RISC Compiler and Simulator

- 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

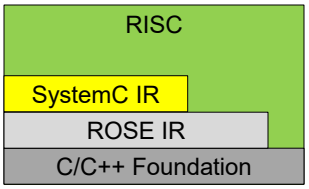
2009 Winner

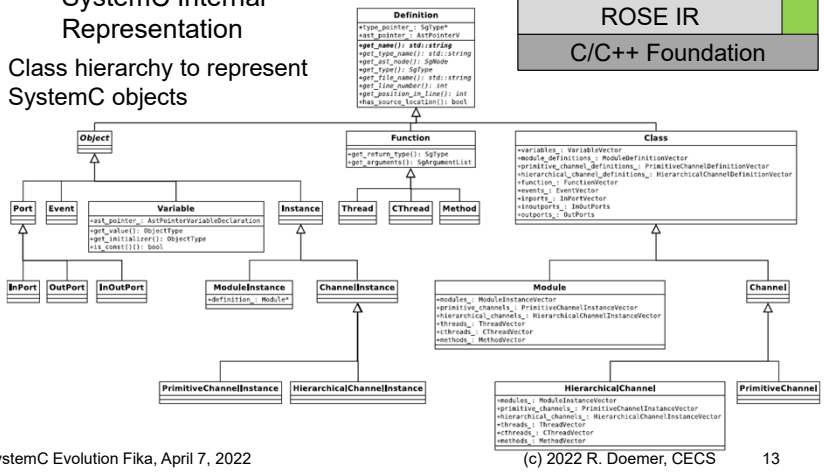
Source:
Lawrence Livermore National Laboratory (LLNL)

12

RISC Compiler and Simulator

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

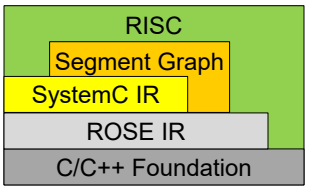


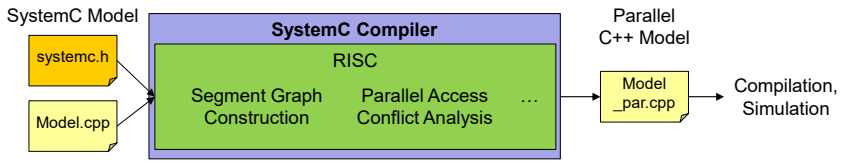


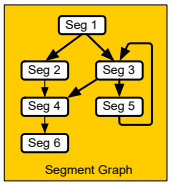
13

RISC Compiler and Simulator

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





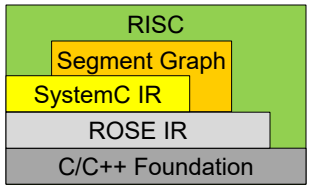


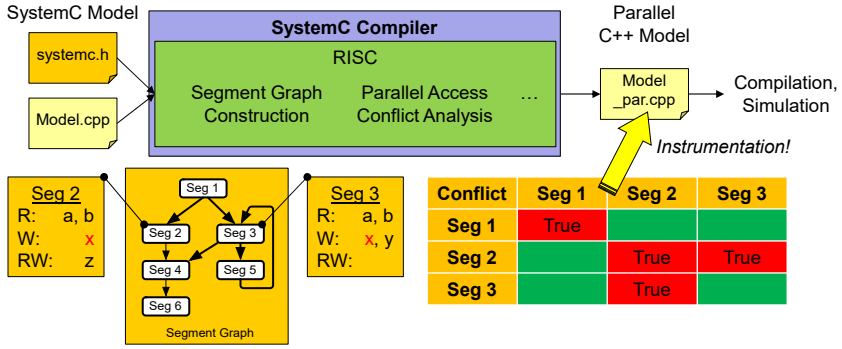
Step 1: Build a Segment Graph

14

RISC Compiler and Simulator

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

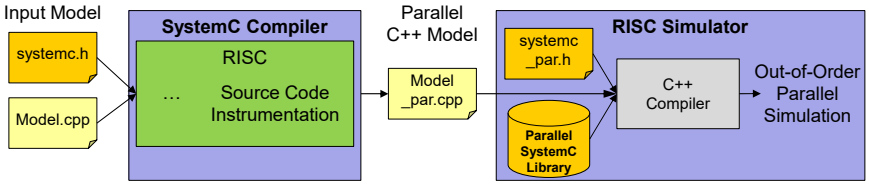




15

RISC Compiler and Simulator

- Compiler and Simulator work hand in hand
  - – Compiler performs conservative conflict analysis
  - – Analysis results are passed to the simulator
  - – Simulator can make safe scheduling decisions quickly
    - ➢ Maximum parallelism
    - ➢ Fast simulation

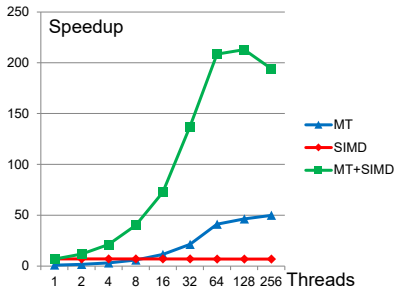


16

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!**

17

RISC Open Source

- RISC Compiler and Simulator, Release V0.6.3
  - ➢ http://www.cecs.uci.edu/~doemer/risc.html#RISC063
    - Installation notes and script: **`INSTALL`**, **`Makefile`**
    - Open source tar ball: **`risc_v0.6.3.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 19-04: **`CECS_TR_19_04.pdf`**
  - ➢ Docker container:

```
bash# docker pull ucirvinelecs/risc063
bash# docker run -it ucirvinelecs/risc063
[dockeruser]# cd demodir
[dockeruser]# make play_demo
```

    - ➢ https://hub.docker.com/r/ucirvinelecs/risc063/

18

Conclusion

- Recoding Infrastructure for SystemC
  - Introduction of a dedicated SystemC compiler
  - Out-of-order parallel simulation on multi- and many-core hosts
  - Maximum compliance with IEEE SystemC semantics
- ➢ Overcomes 7 Obstacles towards Parallel SystemC
  1. Co-Routine Semantics: *Conflicts prevented by compiler*
  2. Simulator State: *Revised SystemC API*
  3. Lack of Thread Safety: *Revised SystemC primitives*
  4. Class sc_channel: *Fixed*
  5. TLM-2.0: *Safety ensured by compiler*
  6. Sequential Mindset: *Not a problem*
  7. Temporal Decoupling: *Future work…*
- Open Source
  - Thanks to Intel Corporation!

Acknowledgments

- For solid work, fruitful discussions, and honest feedback, I would like to thank:
  - My team at UCI
    - Emad Arasteh, Vivek Govindasamy, Yutong Wang
    - Zhongqi Cheng, Daniel Mendoza
    - Guantao Liu, Tim Schmidt
    - Farah Arabi, Aditya Harit, 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!

References (1)

- [FDL'21] E. Arasteh, R. Dömer: *"Improving Parallelism in System Level Models by Assessing PDES Performance"*, FDL, Antibes, France, Sep. 2021
- [Springer'20] 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, Aug. 2020
- [IJPP'20] Z. Cheng, T. Schmidt, R. Dömer: *"Scaled Static Analysis and IP Reuse for Out-of-Order Parallel SystemC Simulation"*, IJPP, Springer, Jun. 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"*, DATE, Grenoble, France, March 2020.
- [ASPDAC'20] Z. Cheng, A. Arasteh, R. Dömer: *"Event Delivery using Prediction for Faster Parallel SystemC Simulation"*, accepted at ASPDAC, Beijing, China, Jan. 2020.
- [CODES+ISSS'19] Z. Cheng, R. Dömer: *"Analyzing Variable Entanglement for Parallel Simulation of SystemC TLM-2.0 Models"*, ACM TECS, vol. 18, no. 5s, article 79, 20 pages, Oct. 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.
- [IESS'19b] E. Arasteh, R. Dömer: *"An Untimed SystemC Model of GoogLeNet"*, Proceedings of IESS, Springer, Friedrichshafen, Germany, Sep. 2019.
- [DVCon'19] D. Mendoza, A. Dingankar, Z. Cheng, R. Dömer: *"Integrating Parallel SystemC Simulation into Simics® Virtual Platform"*, DVCon Europe, Munich, Germany, Oct. 2019.

References (2)

- [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.
- [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'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.