EECS 222: Embedded System Modeling Lecture 20

Rainer Dömer

doemer@uci.edu

The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine

Lecture 20: Overview

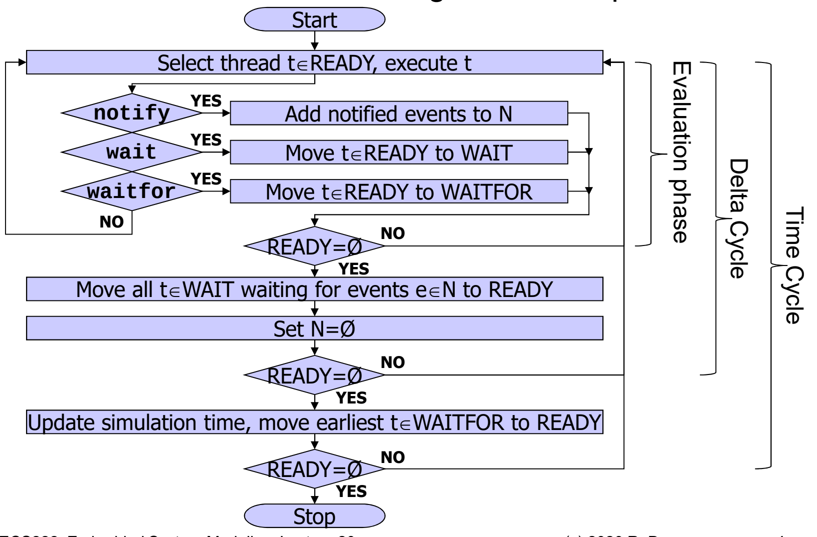
- Discrete Event Simulation Semantics
  - Discrete Event Simulation
  - Parallel Discrete Event Simulation
  - Out-of-Order Parallel Discrete Event Simulation
- Formal Execution Semantics
  - Time-Interval Formalism
- Recoding Infrastructure for SystemC (RISC)
  - Out-of-Order Parallel Simulation for SystemC
  - Experimental results

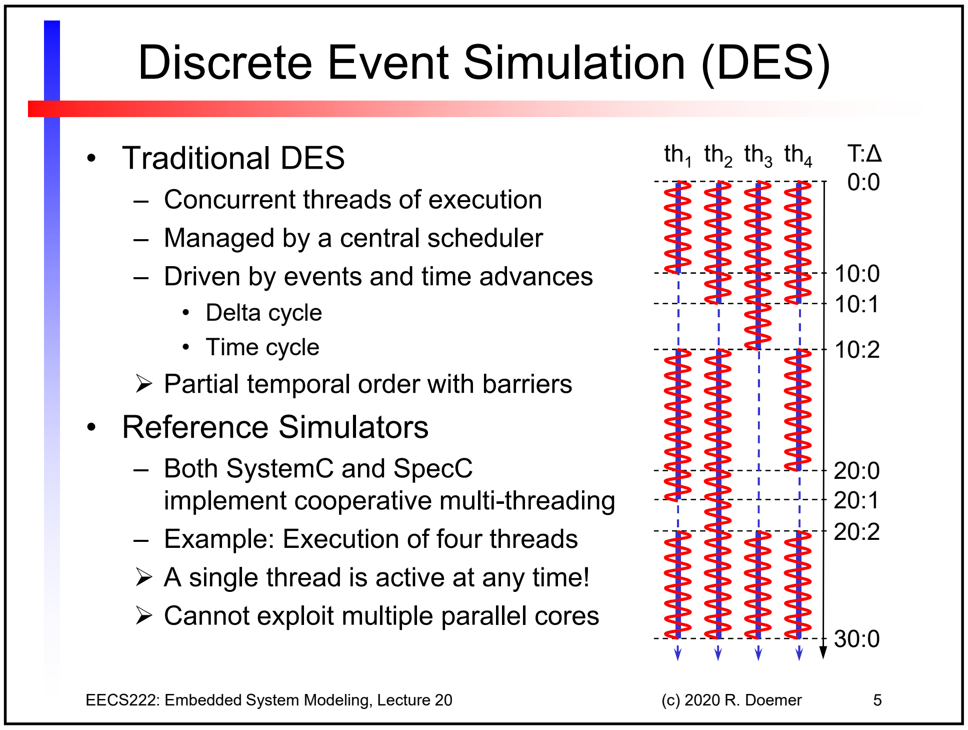
Discrete Event Simulation Semantics

- Discrete Event Simulation Algorithm for SpecC
  – available in LRM (appendix), good for documentation
  ⇒ abstract definition (defines a set of valid implementations)
  ⇒ not general (possibly incomplete)
- Definitions:
  – At any time, each thread t is in one of the following sets:
    - **READY**: set of threads ready to execute (initially root thread)
    - **WAIT**: set of threads suspended by **`wait`** (initially Ø)
    - **WAITFOR**: set of threads suspended by **`waitfor`** (initially Ø)
  – Notified events are stored in a set **N**
    - **`notify e1`** adds event **`e1`** to **N**
    - **`wait e1`** will wakeup when **`e1`** is in **N**
    - Consumption of event e means event e is taken out of **N**
    - Expiration of notified events means **N** is set to Ø

Discrete Event Simulation Semantics

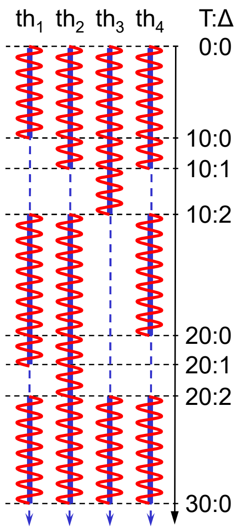
- Discrete Event Simulation Algorithm for SpecC



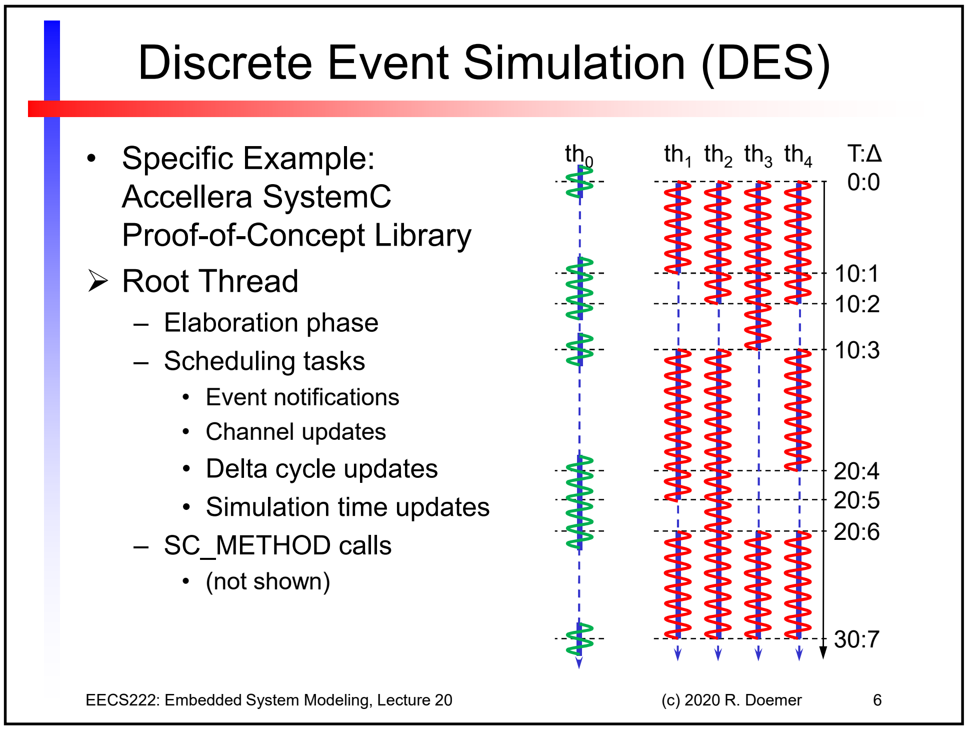


Discrete Event Simulation (DES)

- 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
- Reference Simulators
  - Both SystemC and SpecC implement cooperative multi-threading
  - Example: Execution of four threads
  - ➢ A single thread is active at any time!
  - ➢ Cannot exploit multiple parallel cores

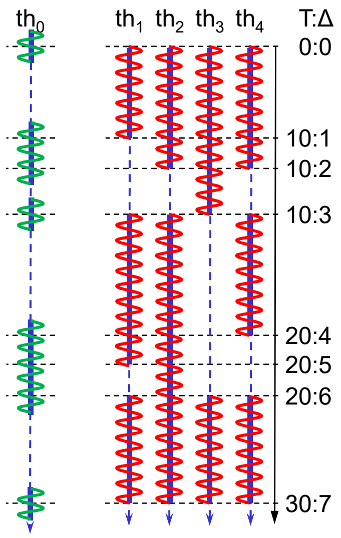


EECS222: Embedded System Modeling, Lecture 20 (c) 2020 R. Doemer 5



Discrete Event Simulation (DES)

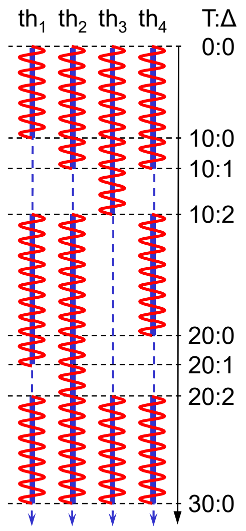
- Specific Example: Accellera SystemC Proof-of-Concept Library
- ➢ Root Thread
  - Elaboration phase
  - Scheduling tasks
    - Event notifications
    - Channel updates
    - Delta cycle updates
    - Simulation time updates
  - SC_METHOD calls
    - (not shown)



EECS222: Embedded System Modeling, Lecture 20 (c) 2020 R. Doemer 6

Discrete Event Simulation (DES)

- ➢ Parallel Simulation!?
- SLDL Execution Semantics
  - SystemC prescribes *Cooperative Multi-Threading*
    - SystemC LRM defines: *“process instances execute without interruption”*
    - ➢ Preemptive scheduling forbidden!
  - SpecC specifies *Preemptive Multi-Threading*
    - SpecC LRM defines: *”preemptive execution”, ”No atomicity is guaranteed”*
    - ➢ Preemptive scheduling assumed!
    - ➢ Need critical regions with mutually exclusive access: Channels!



Formal Execution Semantics

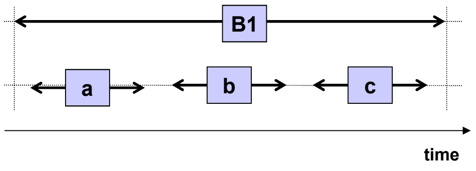
- Examples of Formally Defined Semantics
  1) Time-interval formalism for SpecC
     - Formally defines timed execution semantics
     - Covers sequentiality, concurrency, synchronization
     - Allows reasoning over execution order, dependencies
     - ➢ Discussed in the following slides!
  2) Abstract State Machines (ASM)
     - Completely formal execution semantics
       - wait, notify, notifyone, par, pipe, try-trap-interrupt
       - Operational semantics only (no data types!)
     - Abstract models of SpecC and SystemC match
     - Abstract models closely match VHDL, Verilog
     - ➢ Not discussed in this course

Formal Execution Semantics

- Time-interval formalism
  - Definition of execution semantics of SpecC 2.0
    - sequential execution
    - concurrent execution (semantics of **`par`**)
    - synchronization (semantics of **`notify`**, **`wait`**)
  - Sequential execution

```
behavior B1
{ void main(void)
  { a;
    b;
    c;
  }
};
```

Tstart(B1) <= Tstart(a) < Tend(a) <=
Tstart(b) < Tend(b) <=
Tstart(c) < Tend(c) <= Tend(B1)

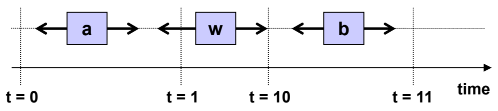


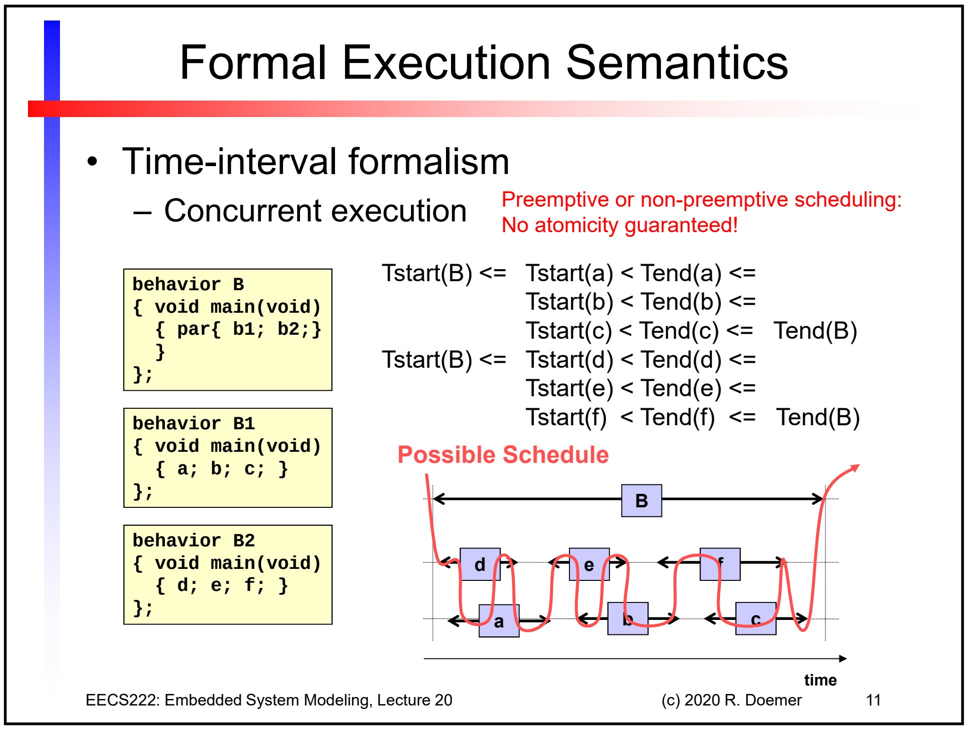
Formal Execution Semantics

- Time-interval formalism
  - Sequential execution
    - waitfor rule:
      - only **`waitfor`** increases simulation time
      - other statements execute in zero simulation time

```
behavior B
{ void main(void)
  { a;
    waitfor 10;
    b;
  }
 };
```

$0 \le$ Tstart(a) < Tend(a) < 1
$0 \le$ Tstart(w) < Tend(w) $= 10$
$10 \le$ Tstart(b) < Tend(b) < 11





Formal Execution Semantics

- Time-interval formalism
  - Concurrent execution

Preemptive or non-preemptive scheduling:
No atomicity guaranteed!

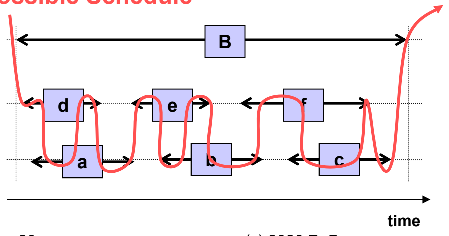
```
behavior B
{ void main(void)
  { par{ b1; b2;}
  }
};
```

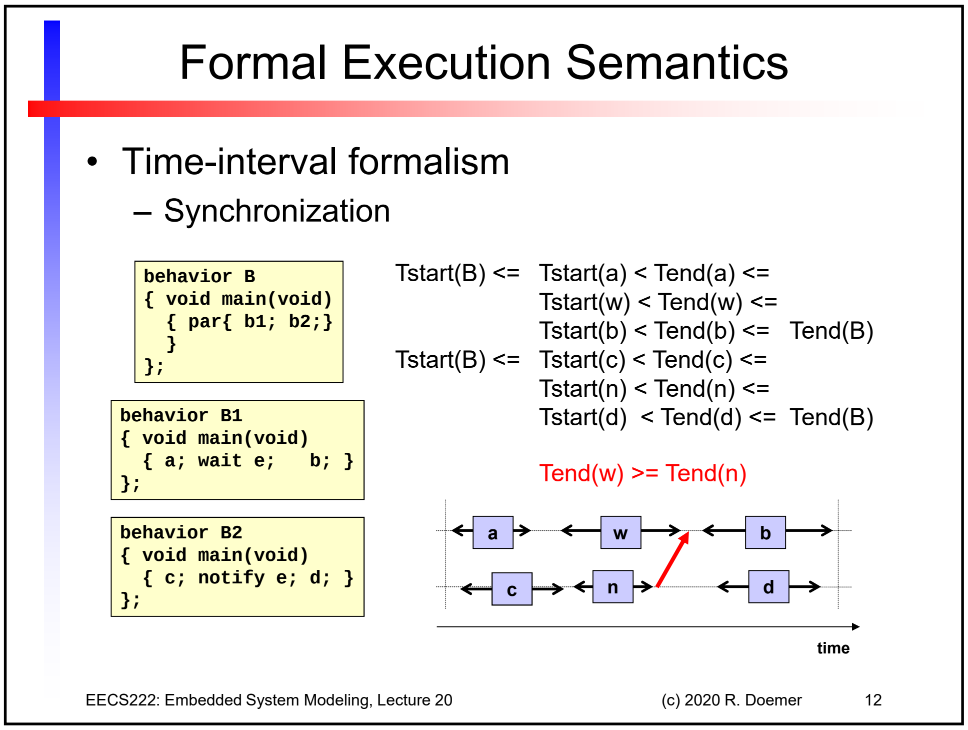
```
behavior B1
{ void main(void)
  { a; b; c; }
};
```

```
behavior B2
{ void main(void)
  { d; e; f; }
};
```

Tstart(B) <= Tstart(a) < Tend(a) <=
Tstart(b) < Tend(b) <=
Tstart(c) < Tend(c) <= Tend(B)
Tstart(B) <= Tstart(d) < Tend(d) <=
Tstart(e) < Tend(e) <=
Tstart(f) < Tend(f) <= Tend(B)

Possible Schedule





Formal Execution Semantics

- Time-interval formalism
  - Synchronization

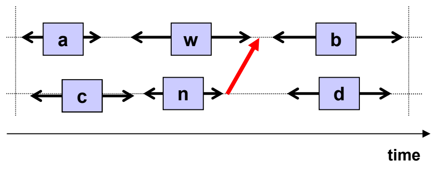
```
behavior B
{ void main(void)
  { par{ b1; b2;}
  }
};
```

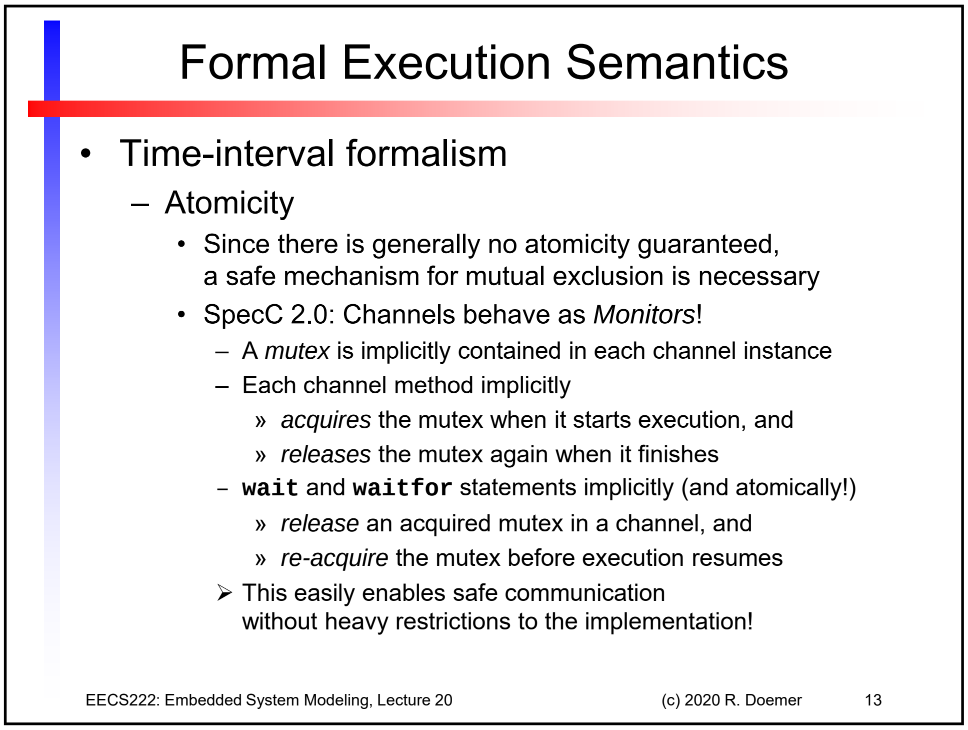
```
behavior B1
{ void main(void)
  { a; wait e;   b; }
};
```

```
behavior B2
{ void main(void)
  { c; notify e; d; }
};
```

Tstart(B) <= Tstart(a) < Tend(a) <=
Tstart(w) < Tend(w) <=
Tstart(b) < Tend(b) <= Tend(B)
Tstart(B) <= Tstart(c) < Tend(c) <=
Tstart(n) < Tend(n) <=
Tstart(d) < Tend(d) <= Tend(B)

Tend(w) >= Tend(n)

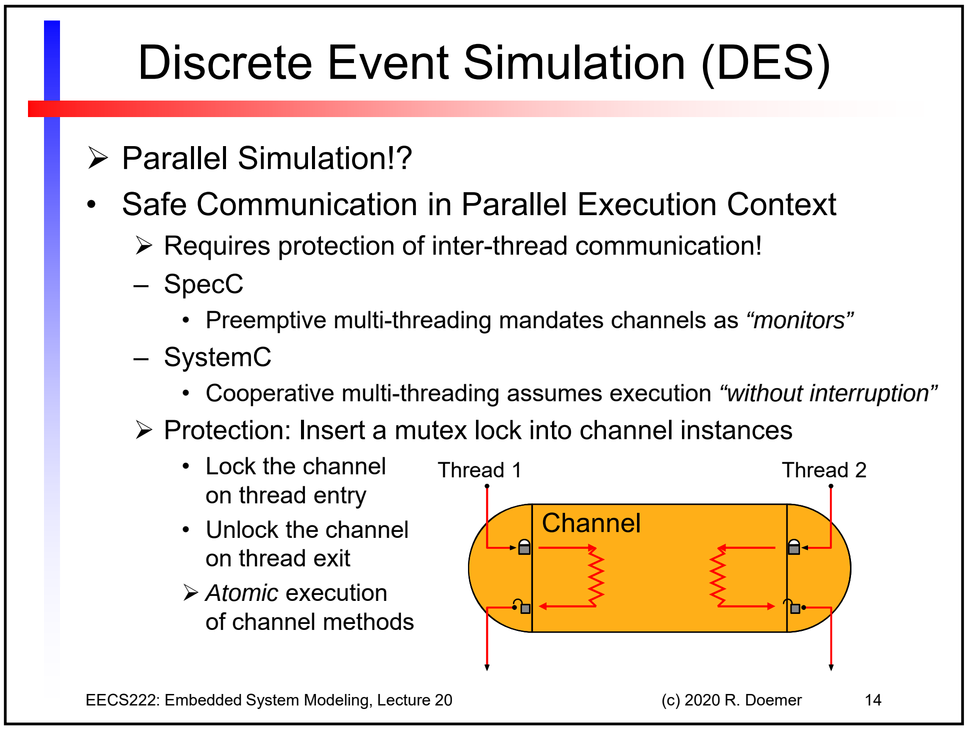




Formal Execution Semantics

- Time-interval formalism
  - Atomicity
    - Since there is generally no atomicity guaranteed,
      a safe mechanism for mutual exclusion is necessary
    - SpecC 2.0: Channels behave as *Monitors*!
      - A *mutex* is implicitly contained in each channel instance
      - Each channel method implicitly
        - » *acquires* the mutex when it starts execution, and
        - » *releases* the mutex again when it finishes
      - **`wait`** and **`waitfor`** statements implicitly (and atomically!)
        - » *release* an acquired mutex in a channel, and
        - » *re-acquire* the mutex before execution resumes
      - ➢ This easily enables safe communication
        without heavy restrictions to the implementation!

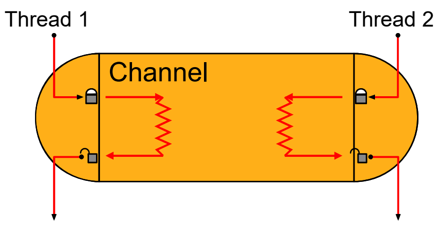
EECS222: Embedded System Modeling, Lecture 20 (c) 2020 R. Doemer 13



Discrete Event Simulation (DES)

➢ Parallel Simulation!?

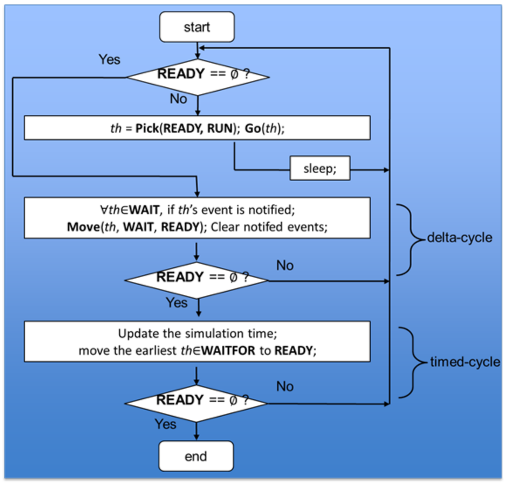
- Safe Communication in Parallel Execution Context
  - ➢ Requires protection of inter-thread communication!
  - SpecC
    - Preemptive multi-threading mandates channels as *"monitors"*
  - SystemC
    - Cooperative multi-threading assumes execution *"without interruption"*
  - ➢ Protection: Insert a mutex lock into channel instances
    - Lock the channel on thread entry
    - Unlock the channel on thread exit
    - ➢ *Atomic* execution of channel methods



EECS222: Embedded System Modeling, Lecture 20 (c) 2020 R. Doemer 14

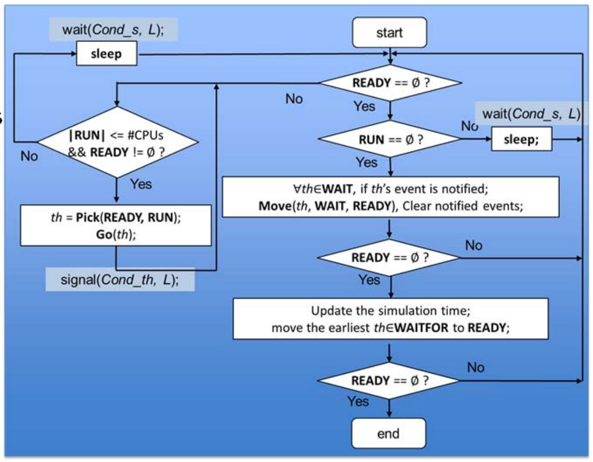
Parallel Discrete Event Simulation (PDES)

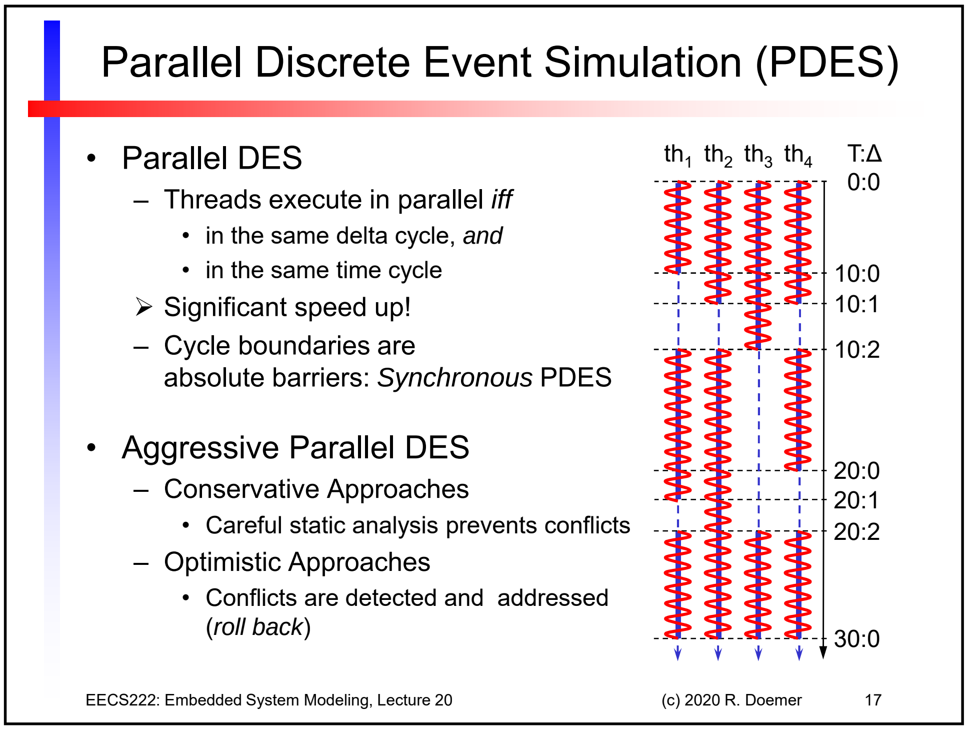
- Review: Sequential DES Algorithm
  - Active Threads are managed in READY queue
  - Simulation progress
    - Delta cycle
    - Time cycle
  - ➢ Scheduler picks a *single* thread and executes it


start
READY == ∅ ?
Yes
No
th = Pick(READY, RUN); Go(th);
sleep;
∀th∈WAIT, if th's event is notified;
Move(th, WAIT, READY); Clear notifed events;
delta-cycle
READY == ∅ ?
No
Yes
Update the simulation time;
move the earliest th∈WAITFOR to READY;
timed-cycle
READY == ∅ ?
No
Yes
end

Parallel Discrete Event Simulation (PDES)

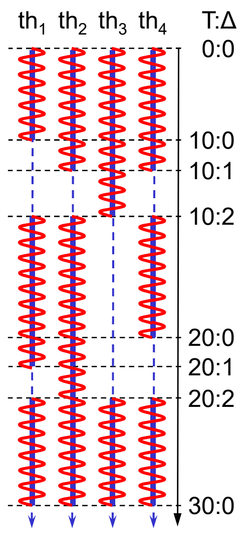
- Parallel DES Algorithm
  - Active threads are managed in READY queue
  - Simulation progress
    - Delta cycle
    - Time cycle
  - ➢ Scheduler *picks N threads* and executes them *in parallel*
  - ➢ *N* = number of available CPU cores

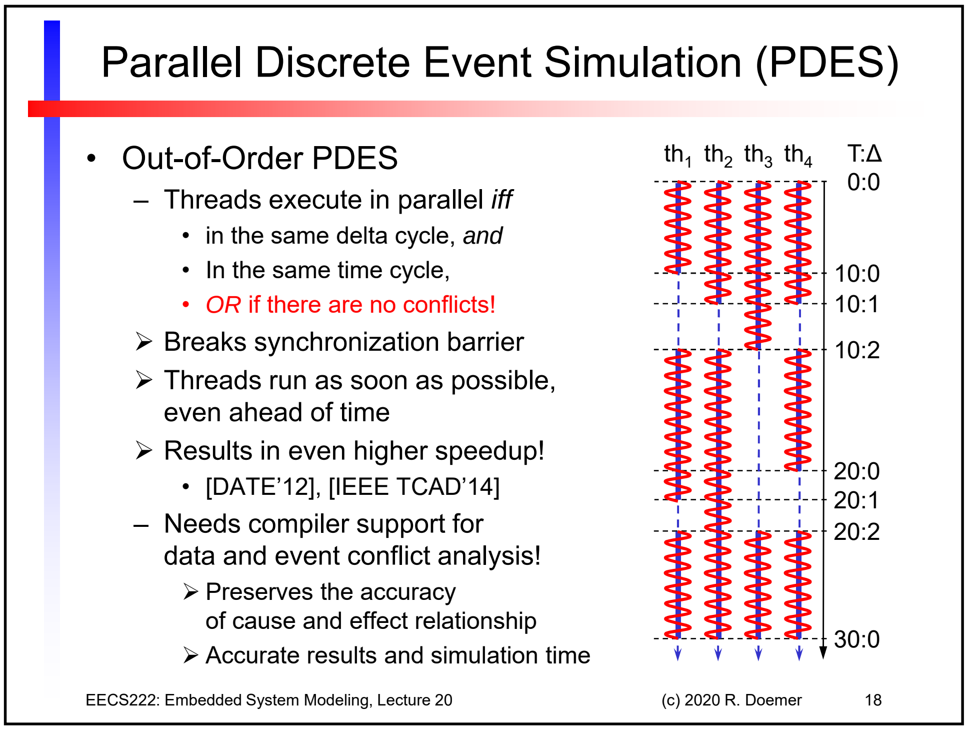

wait(Cond_s, L);
sleep
start
READY == ∅ ?
No
Yes
|RUN| <= #CPUs && READY != ∅ ?
No
Yes
th = Pick(READY, RUN); Go(th);
signal(Cond_th, L);
RUN == ∅ ?
No
wait(Cond_s, L)
sleep;
Yes
∀th∈WAIT, if th's event is notified;
Move(th, WAIT, READY), Clear notified events;
READY == ∅ ?
No
Yes
Update the simulation time;
move the earliest th∈WAITFOR to READY;
READY == ∅ ?
No
Yes
end



Parallel Discrete Event Simulation (PDES)

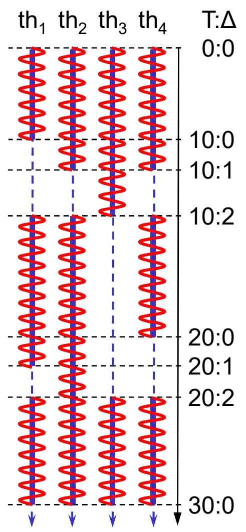
- Parallel DES
  - Threads execute in parallel *iff*
    - in the same delta cycle, *and*
    - in the same time cycle
  - ➢ Significant speed up!
  - Cycle boundaries are absolute barriers: *Synchronous* PDES
- Aggressive Parallel DES
  - Conservative Approaches
    - Careful static analysis prevents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

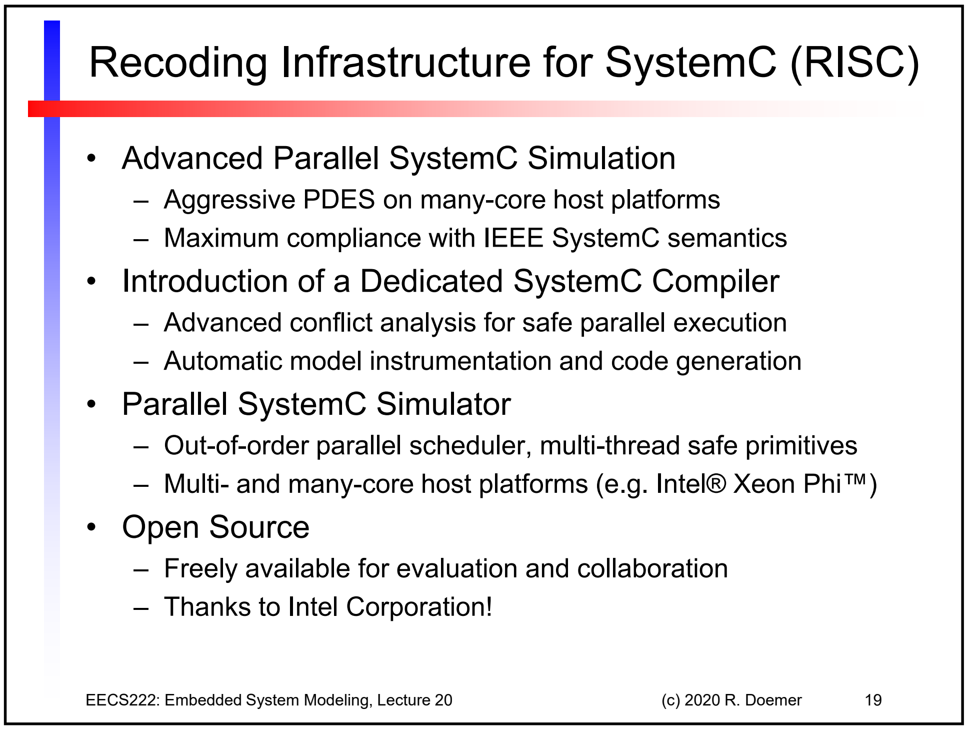




Parallel Discrete Event Simulation (PDES)

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

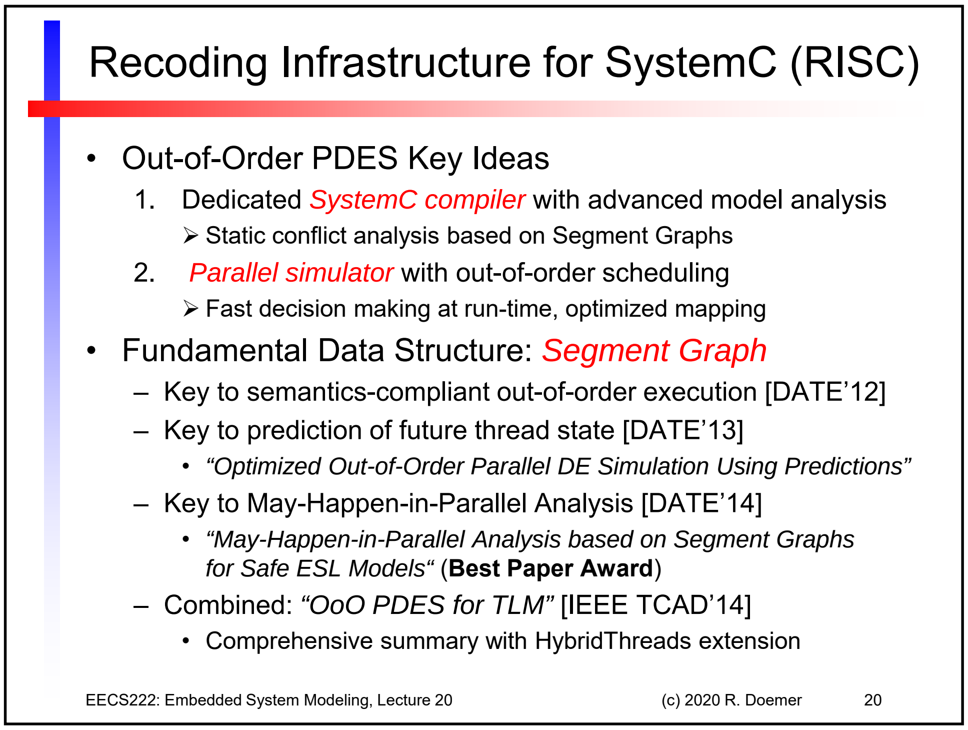




Recoding Infrastructure for SystemC (RISC)

- 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!

EECS222: Embedded System Modeling, Lecture 20 (c) 2020 R. Doemer 19



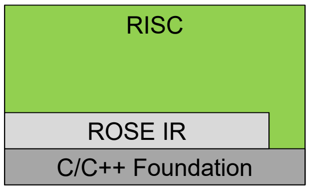
Recoding Infrastructure for SystemC (RISC)

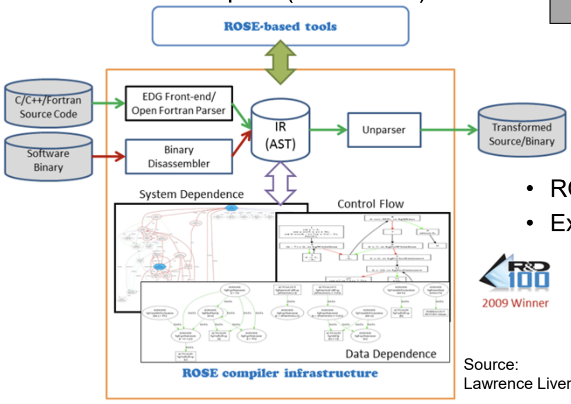
- Out-of-Order PDES Key Ideas
  1. Dedicated *SystemC compiler* with advanced model analysis
     - ➢ Static conflict analysis based on Segment Graphs
  2. *Parallel simulator* with out-of-order scheduling
     - ➢ Fast decision making at run-time, optimized mapping
- Fundamental Data Structure: *Segment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

EECS222: Embedded System Modeling, Lecture 20 (c) 2020 R. Doemer 20

RISC: Dedicated SystemC Compiler

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




ROSE-based tools
C/C++/Fortran Source Code
Software Binary
EDG Front-end/ Open Fortran Parser
Binary Disassembler
IR (AST)
Unparser
Transformed Source/Binary
System Dependence
Control Flow
Data Dependence
ROSE compiler infrastructure

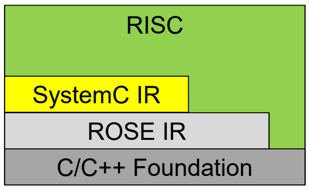
- ROSE Internal Representation
- Explicit support for
  - Source code analysis
  - Source-to-source transformations

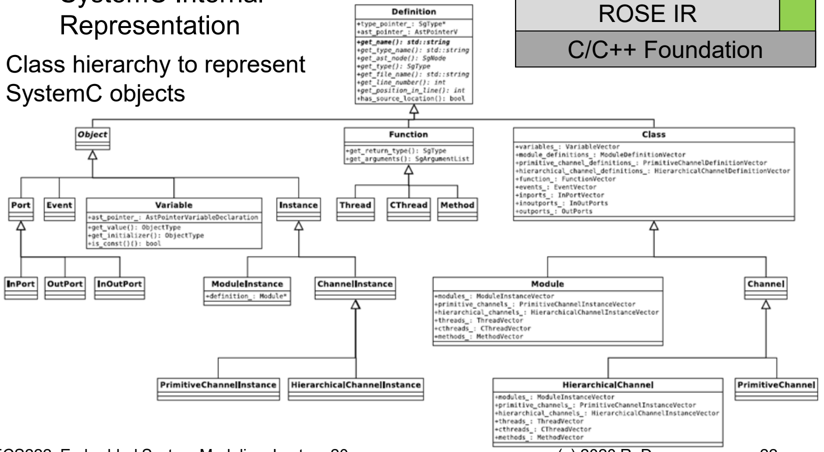
2009 Winner

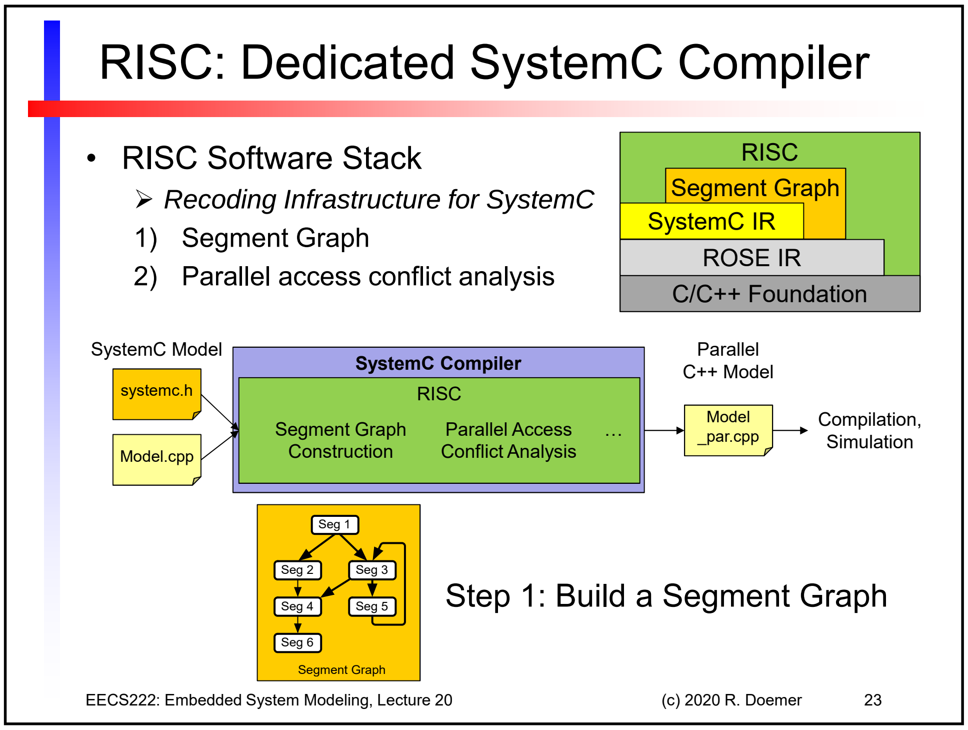
Source:
Lawrence Livermore National Laboratory (LLNL)

RISC: Dedicated SystemC Compiler

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

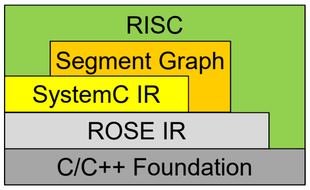


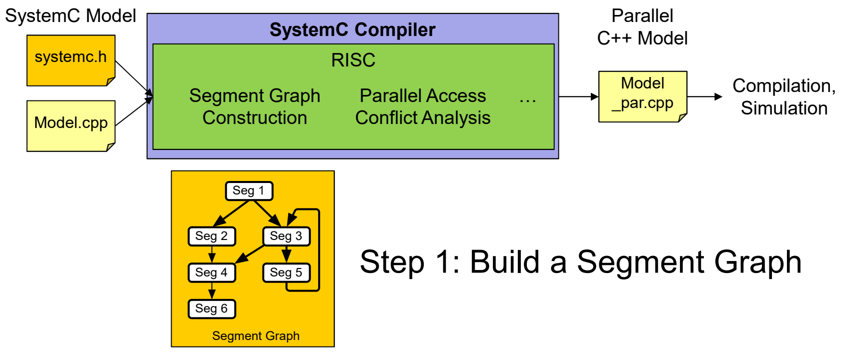




RISC: Dedicated SystemC Compiler

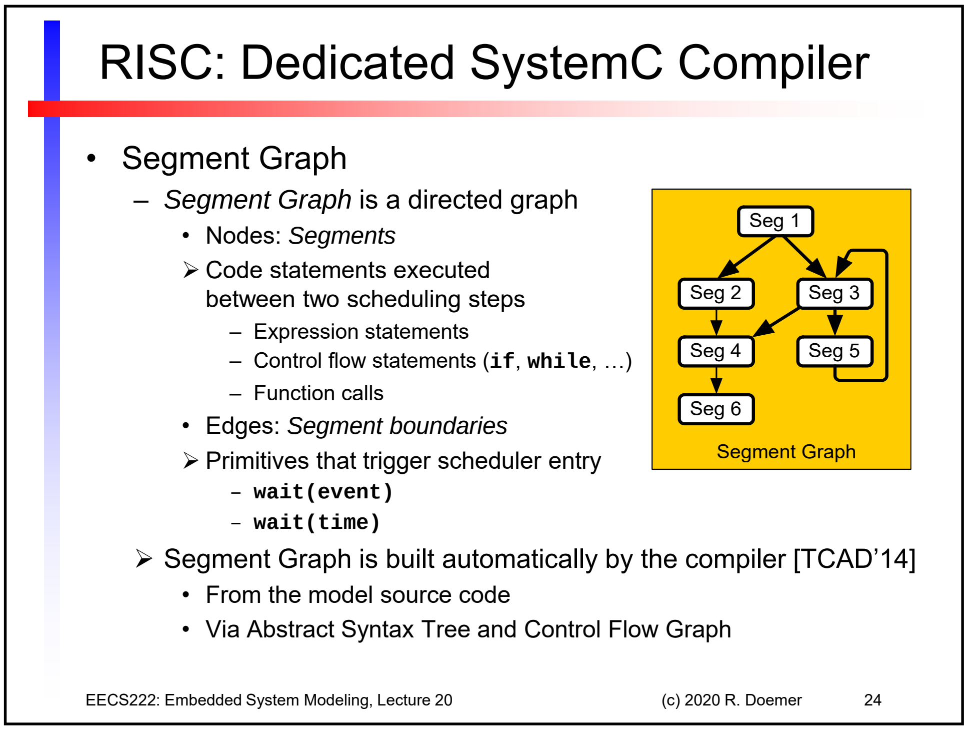
- RISC Software Stack
  - ➢ *Recoding Infrastructure for SystemC*
  - 1) Segment Graph
  - 2) Parallel access conflict analysis





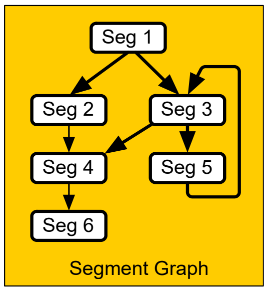
Step 1: Build a Segment Graph

EECS222: Embedded System Modeling, Lecture 20 (c) 2020 R. Doemer 23



RISC: Dedicated SystemC Compiler

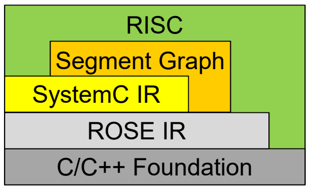
- Segment Graph
  - – *Segment Graph* is a directed graph
    - Nodes: *Segments*
    - ➢ Code statements executed between two scheduling steps
      - – Expression statements
      - – Control flow statements (**`if`**, **`while`**, …)
      - – Function calls
    - Edges: *Segment boundaries*
    - ➢ 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

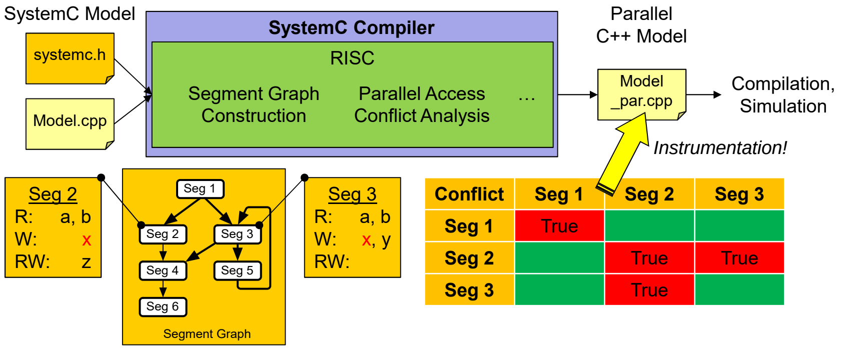


EECS222: Embedded System Modeling, Lecture 20 (c) 2020 R. Doemer 24

RISC: Dedicated SystemC Compiler

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

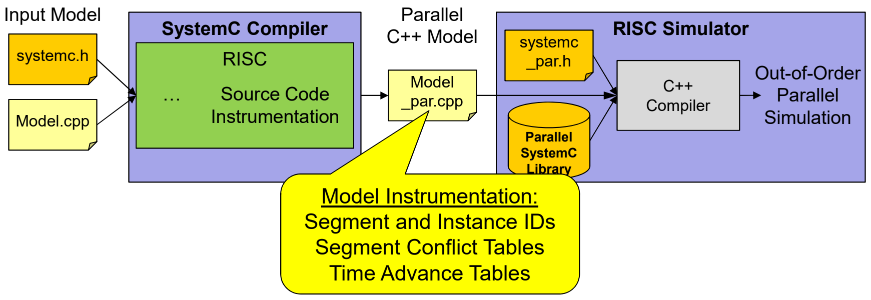




Conflict	Seg 1	Seg 2	Seg 3
Seg 1	True	🟩	🟩
Seg 2	🟩	True	True
Seg 3	🟩	True	🟩

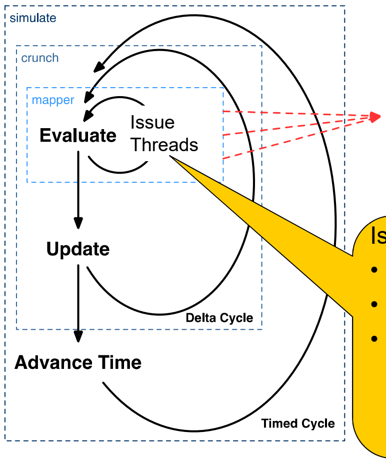
RISC: Compiler and Simulator

- 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



RISC: Parallel SystemC Simulator

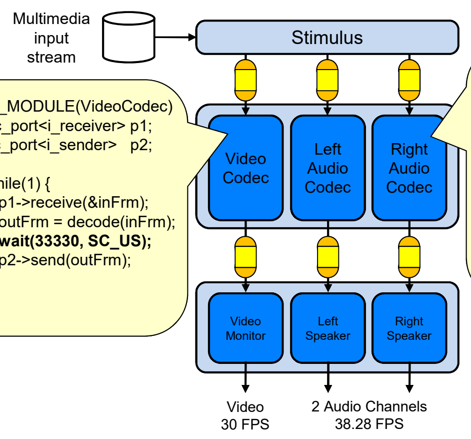
- Simulator kernel with Out-of-Order Parallel Scheduler
  – Conceptual OoO PDES execution



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

RISC: Experiments and Results

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



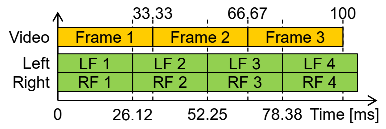
```
SC_MODULE(VideoCodec)
{ sc_port<i_receiver> p1;
  sc_port<i_sender>   p2;
  …
  while(1) {
    p1->receive(&inFrm);
    outFrm = decode(inFrm);
    wait(33330, SC_US);
    p2->send(outFrm);
  }
};
```

```
SC_MODULE(AudioCodec)
{ sc_port<i_receiver> p1;
  sc_port<i_sender>   p2;
  …
  while(1) {
    p1->receive(&inFrm);
    outFrm = decode(inFrm);
    wait(26120, SC_US);
    p2->send(outFrm);
  }
};
```

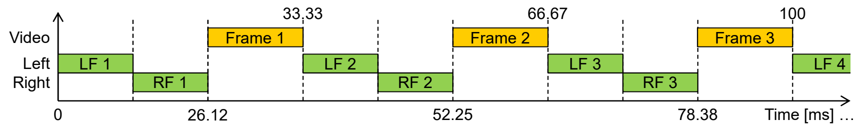
RISC: Experiments and Results

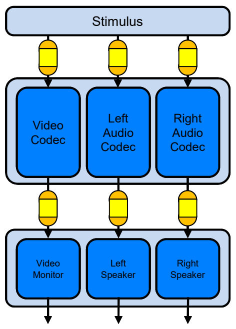
- DVD Player Example
  - Parallel video and audio decoding with different frame rates

1. Real time schedule: fully parallel



2. Reference simulator schedule (DES)

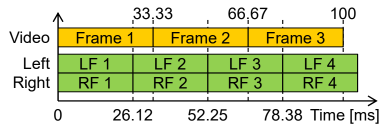




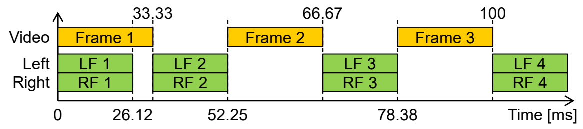
RISC: Experiments and Results

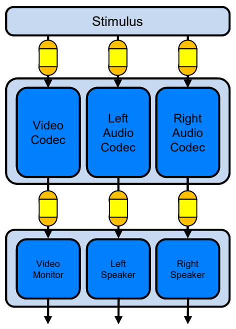
- DVD Player Example
  - Parallel video and audio decoding with different frame rates

1. Real time schedule: fully parallel



3. Synchronous parallel schedule (PDES)

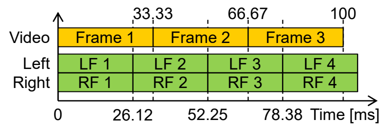




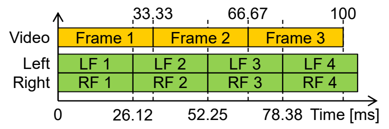
RISC: Experiments and Results

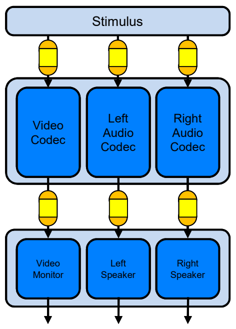
- DVD Player Example
  - Parallel video and audio decoding with different frame rates

  1. Real time schedule: fully parallel



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

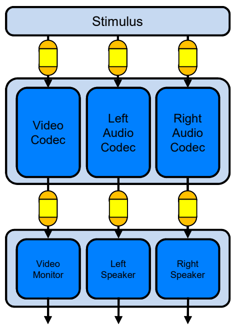




RISC: Experiments and Results

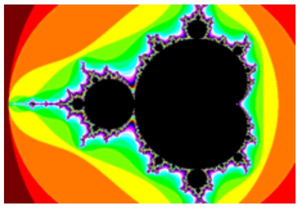
- DVD Player Example
  - Parallel video and audio decoding with different frame rates
- Simulator Run Times
  - 4-core Intel® Xeon® CPU at 3.4 GHz
  - RISC v0.2.1, Posix-threads

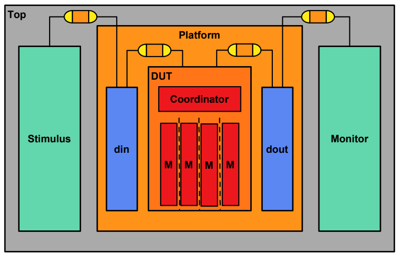
		DES	PDES	OoO PDES
10 sec stream	Run Time	6.98 s	4.67 s	2.94 s
	CPU Load	97%	145%	238%
	Speedup	1 x	1.49 x	2.37 x
100 sec stream	Run Time	68.21 s	45.91 s	28.13 s
	CPU Load	100%	149%	251%
	Speedup	1 x	1.49 x	2.42 x



RISC: Experiments and Results

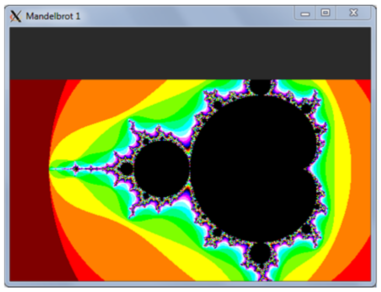
- Mandelbrot Renderer (Graphics Pipeline Application)
  - Mandelbrot Set
    - Mathematical set of points in complex plane
      - Two-dimensional fractal shape
    - High computation load
      - Recursive/iterative function
    - Embarrassingly parallel
      - Parallelism at pixel level
  - SystemC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

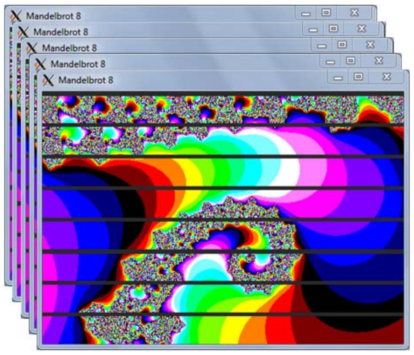


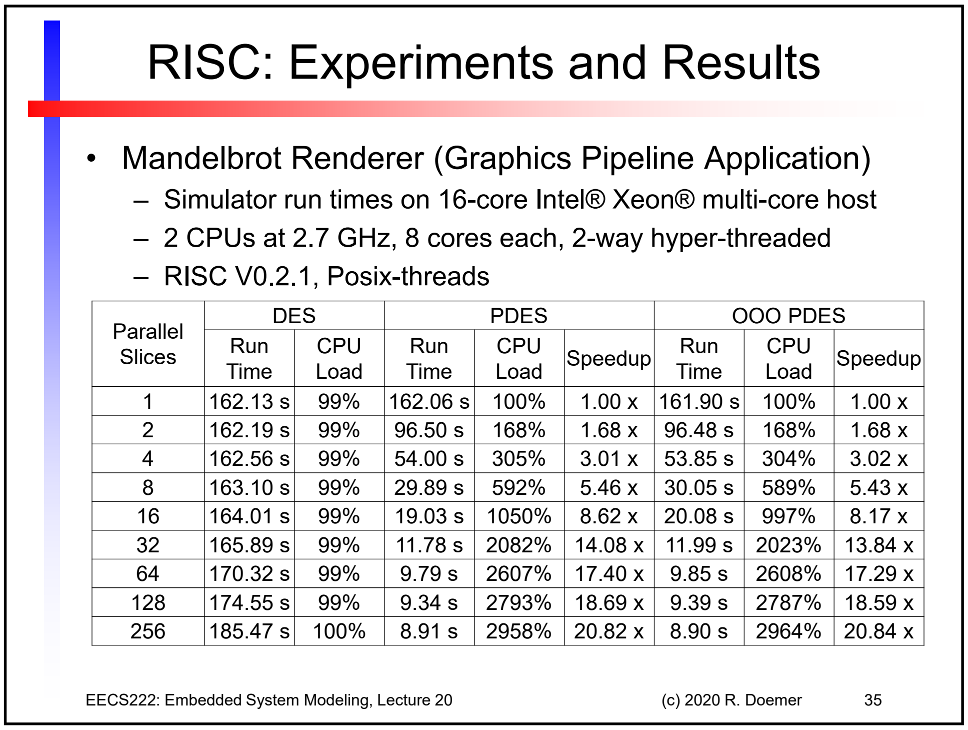


RISC: Experiments and Results

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





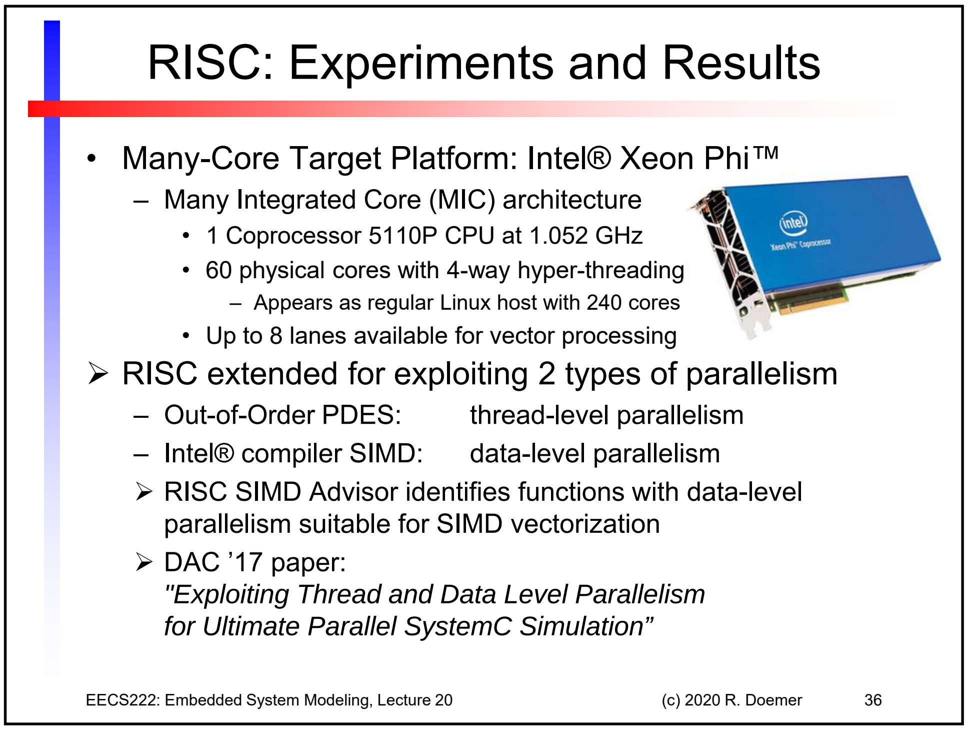


RISC: Experiments and Results

- 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

EECS222: Embedded System Modeling, Lecture 20 (c) 2020 R. Doemer 35



RISC: 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

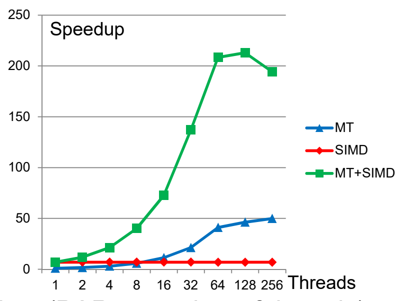
➢ RISC extended for exploiting 2 types of parallelism
  - Out-of-Order PDES: thread-level parallelism
  - Intel® compiler SIMD: data-level parallelism
  - ➢ RISC SIMD Advisor identifies functions with data-level parallelism suitable for SIMD vectorization
  - ➢ DAC ’17 paper:
    "Exploiting Thread and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

EECS222: Embedded System Modeling, Lecture 20 (c) 2020 R. Doemer 36

RISC: 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#RISC060
    - Installation notes and script: **`INSTALL`**, **`Makefile`**
    - Open source tar ball: **`risc_v0.6.0.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/risc060
bash# docker run -it ucirvinelecs/risc060
[dockeruser]# cd demodir
[dockeruser]# make test
```

  ➢ https://hub.docker.com/r/ucirvinelecs/risc060/