Out-of-Order Parallel Simulation of SystemC Models on Many-Core Platforms

Presentation at UNSW, 2 March 2016

Rainer Dömer, Guantao Liu, Tim Schmidt
{doemer,guantaol,schmidtt}@uci.edu

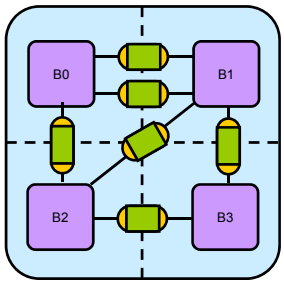
Center for Embedded and Cyber-Physical Systems
University of California, Irvine

UCIrvine
University of California, Irvine

Project Context: ESL Design

- Electronic System Level Models
  - Abstract description of a complete system
  - Hardware + Software
- Key Concepts in System Modeling
  - Explicit Structure
    - Block diagram structure
    - Connectivity through ports
  - Explicit Hierarchy
    - System composed of components
  - Explicit Concurrency
    - Potential for parallel execution
    - Potential for pipelined execution
  - Explicit Communication and Computation
    - Modules
    - Channels and Interfaces

SystemC Model



Project Context: ESL Design

- Model Validation through Simulation!
  - Efficient system-level simulation is critical
    - Fast, and
    - Accurate!
  - Complexity of system models grows constantly
    - Need for speed!
- Parallel Simulation!
  - Parallelism explicitly specified in model
    - System-level Description Language (SLDL)
      - SystemC [Groetker et. al, 2002): `SC_THREAD, SC_METHOD`
      - SpecC [Gajski et. al, 2000]: `par { }, pipe { }`
  - Parallel processing available in standard PCs
    - Multi-core host PCs readily available
    - Many-core technology is arriving

Project Context: Related Work

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. [IEEED&T'11]
•Yun et al. [TCAD'12]

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

Project with Intel: Key Points

- Advanced Parallel SystemC Simulation
  - Out-of-Order PDES on many-core host platforms
  - Maximum compliance with current execution semantics
  - Support for parallel execution of virtual platforms
- Introduction of a Dedicated SystemC Compiler
  - Recoding Infrastructure for SystemC (RISC)
  - Advanced static analysis for parallel execution
  - Model instrumentation and code generation
- Parallel SystemC Core Library
  - Out-of-order parallel scheduler, multi-thread safe primitives
  - Many-core target platform (e.g. Intel® Xeon Phi™)
- Open Source
  - Collaboration with Accellera SystemC Language WG

Outline

- Advanced Parallel SystemC Simulation
  - Traditional Discrete Event Simulation (DES)
  - Parallel Discrete Event Simulation (PDES)
  - Out-of-Order Parallel Discrete Event Simulation (OoO PDES)
- Project Overview
  - SystemC Compiler and Parallel Simulation Kernel
    - Recoding Infrastructure for SystemC (RISC), Segment Graph
    - Out-of-order parallel thread scheduling, many-core platforms
  - Demo and Experimental Results
    - Embedded application: Conceptual DVD player
    - Highly parallel application: Mandelbrot renderer
  - Prototype Implementation
    - Open source alpha release available
- Concluding Remarks

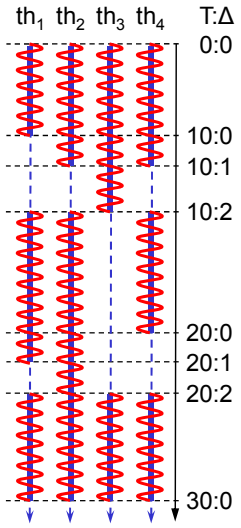
Out-of-Order PDES Technology

- ➢ *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*!

Out-of-Order Parallel SystemC, UNSW, 2 March 2016 (c) 2016 R. Doemer, CECS 7

Discrete Event Simulation (DES)

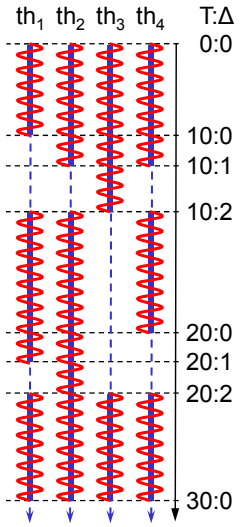
- Traditional DES
  - Concurrent threads of execution
  - Managed by a central scheduler
  - Driven by events and time advances
    - Delta-cycle
    - Time-cycle
  - ➢ Partial temporal order with barriers
- Standard Simulator
  - SystemC reference simulator uses cooperative multi-threading
  - ➢ A single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



Out-of-Order Parallel SystemC, UNSW, 2 March 2016 (c) 2016 R. Doemer, CECS 8

Parallel Discrete Event Simulation (PDES)

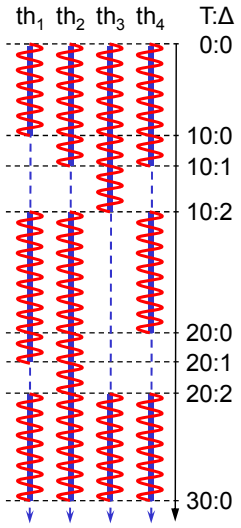
- Parallel DES
  - Threads execute in parallel *iff*
    - in the same delta cycle, *and*
    - in the same time cycle
  - ➢ Significant speed up!
  - *Synchronous* PDES: Cycle boundaries are *absolute barriers*!
- Aggressive Parallel DES
  - Conservative Approaches
    - Careful static analysis prevents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



Out-of-Order Parallel SystemC, UNSW, 2 March 2016 (c) 2016 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



Out-of-Order Parallel SystemC, UNSW, 2 March 2016 (c) 2016 R. Doemer, CECS 10

Out-of-Order PDES Technology

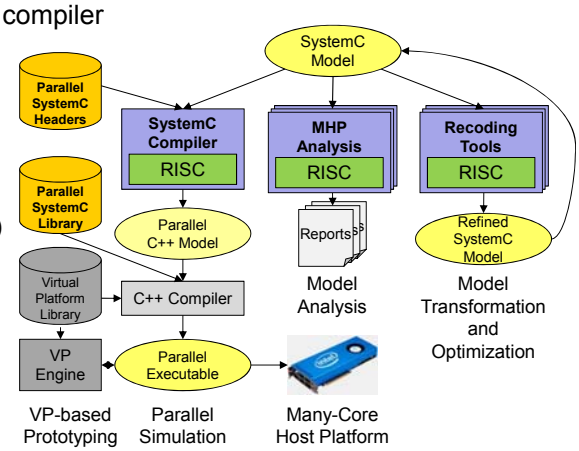
- OoO PDES Key Ideas
  1. Dedicated *SystemC compiler* with advanced model analysis
     - ➢ Static conflict analysis based on Segment Graphs
  2. *Parallel simulator* with out-of-order scheduling on many cores
     - ➢ Fast decision making at run-time, optimized mapping
- Fundamental Data Structure: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

Out-of-Order Parallel SystemC, UNSW, 2 March 2016 (c) 2016 R. Doemer, CECS 11

Project Overview and Tool Flow

- Research and Development Tasks

1) Dedicated SystemC compiler (RISC compiler)
2) Parallel SystemC simulator
3) Performance tuning for many-core hosts
4) Virtual Platform (VP) integration
5) Model analysis (may-happen-in-parallel, MHP)
6) Model recoding, transformation and optimization



Out-of-Order Parallel SystemC, UNSW, 2 March 2016 (c) 2016 R. Doemer, CECS 12

Project Status after Year 1 of 3

- Research and Development Tasks Completed

Y1 Dedicated SystemC compiler ☑ (RISC compiler)

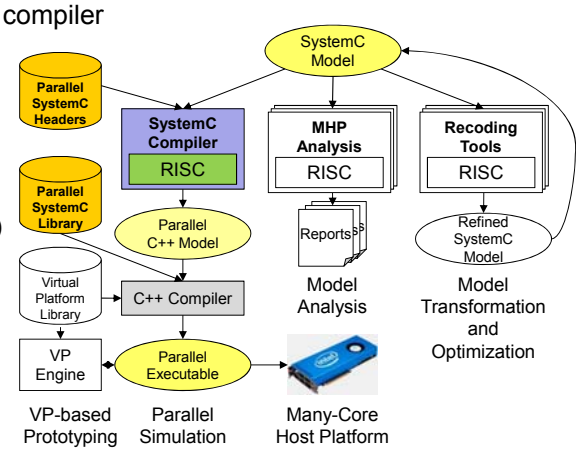
Y1 Parallel SystemC ☑ simulator

Y1 Performance tuning ☑ for many-core hosts

Y2 Virtual Platform (VP) ☐ integration

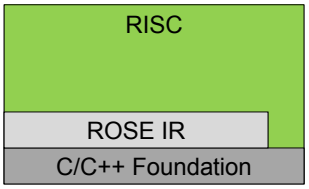
Y3 Model analysis ☐ (may-happen-in-parallel, MHP)

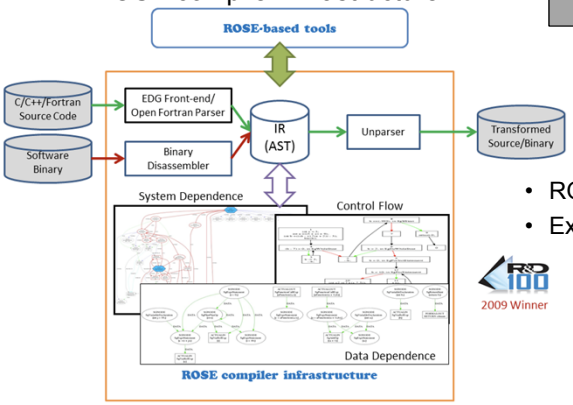
Y3 Model recoding, ☐ transformation and optimization



Dedicated SystemC Compiler

- RISC Software Stack
  - ➢ *Recoding Infrastructure for SystemC*
  - – C/C++ foundation
  - – ROSE compiler infrastructure



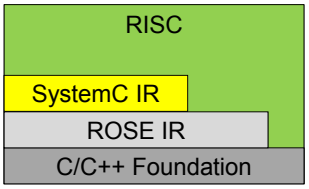


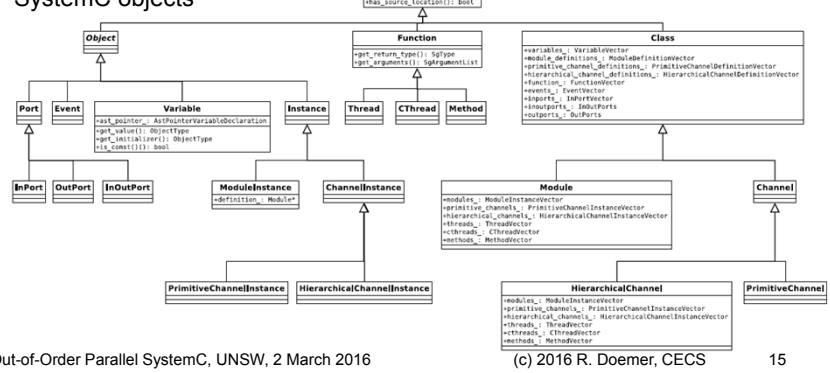
- ROSE Internal Representation
- Explicit support for
  - Source code analysis
  - Source-to-source transformations

2009 Winner

Dedicated SystemC Compiler

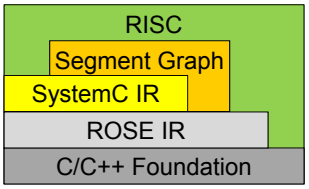
- RISC Software Stack
  - ➢ *Recoding Infrastructure for SystemC*
  - – SystemC Internal Representation
- Class hierarchy to represent SystemC objects

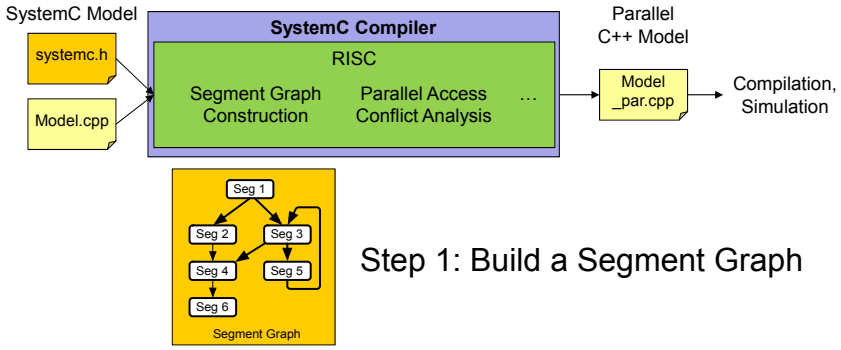




Dedicated SystemC Compiler

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

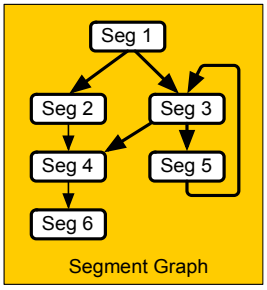




Step 1: Build a Segment Graph

Dedicated SystemC Compiler

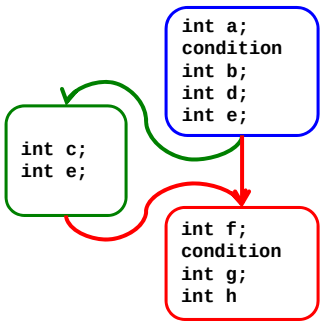
- Segment Graph
  - *Segment Graph* is a directed graph
    - Nodes: *Segments*
    - ➢ Code statements executed between two scheduling steps
      - Expression statements
      - Control flow statements (if, while, …)
      - Function calls
    - Edges: *Segment boundaries*
    - ➢ Primitives that trigger scheduler entry
      - **`wait(event)`**
      - **`wait(time)`**
  - ➢ Segment Graph can be constructed statically by the compiler from the model source code
    - (see example on next slide)



Dedicated SystemC Compiler

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

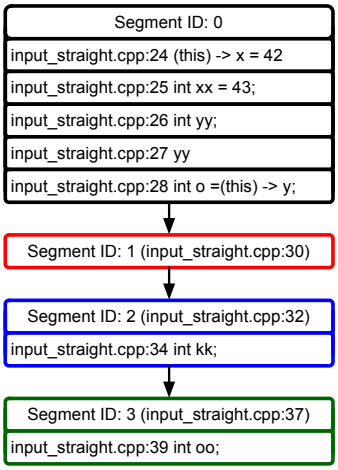
```
int a;
if(cond) {
    int b;
    wait(1);
    int c;
} else {
    int d;
}
int e;
wait(2);
int f;
while(cond) {
    int g;
}
int h;
```



Dedicated SystemC Compiler

- Segment Graph Construction:
  - Support for straight-line code

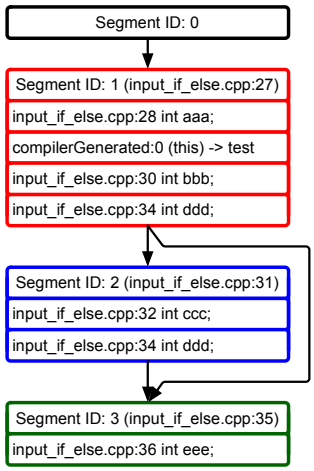
```
void straight()
{
     x = 42;
     int xx = 43;
     int yy;
     yy;
     int o = y;

     wait(10, SC_NS);

     wait();

     int kk;

     wait();

     int oo;
}
```



Dedicated SystemC Compiler

- Segment Graph Construction:
  - Support for conditional statements
    - ➢ **`if`**, **`if-else`**, **`switch-case`** (with **`break`**)

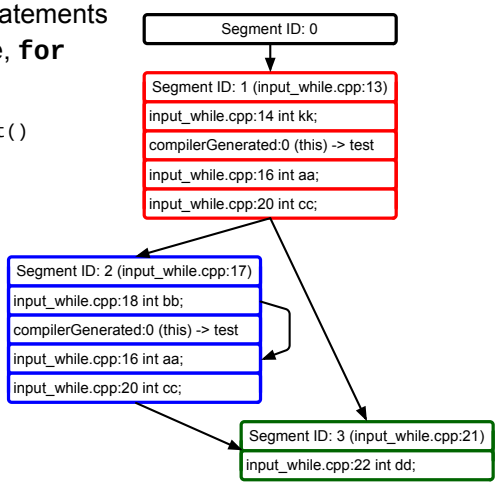
```
void if_statement()
{
     wait();
     int aaa;
     if(test) {
         int bbb;
         wait();
         int ccc;
     }
     int ddd;
     wait();
     int eee;
}
```



Dedicated SystemC Compiler

- Segment Graph Construction:
  - Support for loop statements
    - **while**, **do-while**, **for**

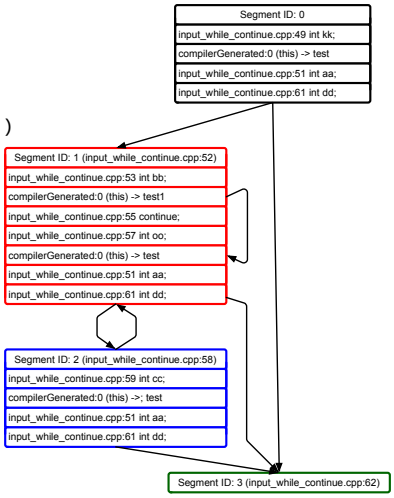
```
void while_statement()
{
     wait();
     int kk;
     while(test) {
         int aa;
         wait();
         int bb;
     }
     int cc;
     wait();
     int dd;
}
```



Dedicated SystemC Compiler

- Segment Graph Construction:
  - Support for loop statements
    - **while**, **do-while**, **for** (with **break**, **continue**)

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

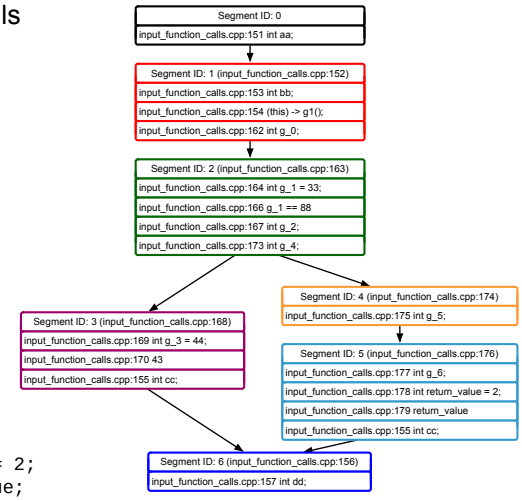


Dedicated SystemC Compiler

- Segment Graph Construction:
  - Support for function calls
    - ➢ **f(x), return**

```
void f()
{
  int aa;
  wait();
  int bb;
  g1();
  int cc;
  wait();
  int dd;
}
```

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

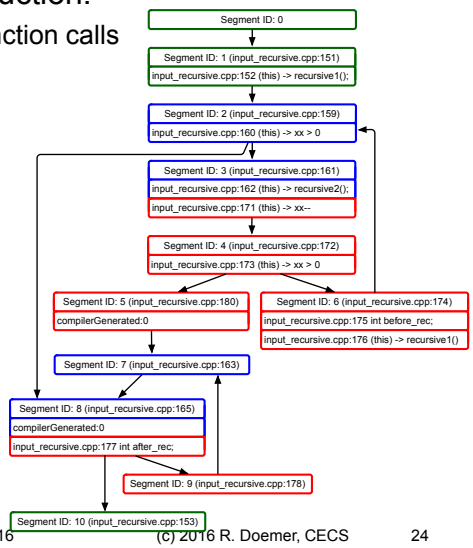


Dedicated SystemC Compiler

- Segment Graph Construction:
  - Support for *recursive* function calls
    - ➢ Direct, indirect recursion

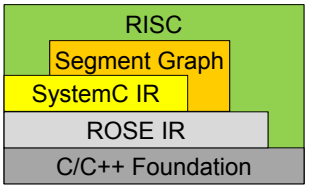
```
void main()
{ wait();
  f();
  wait();
}

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

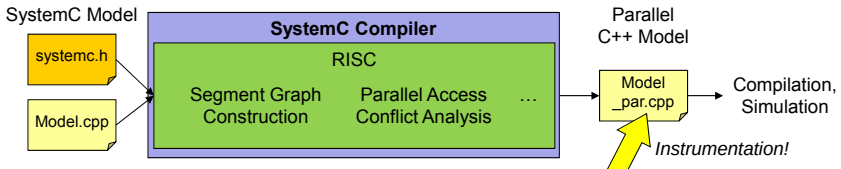
```
void f()
{ wait();
  if(xx>0) {
    wait();
    g();
    wait();
  }
  wait();
  return;
}
```

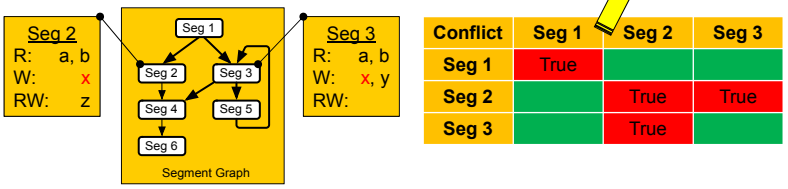


Dedicated SystemC Compiler

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



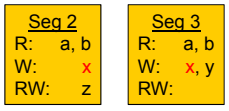




Conflict	Seg 1	Seg 2	Seg 3
Seg 1	True		
Seg 2		True	True
Seg 3		True	

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



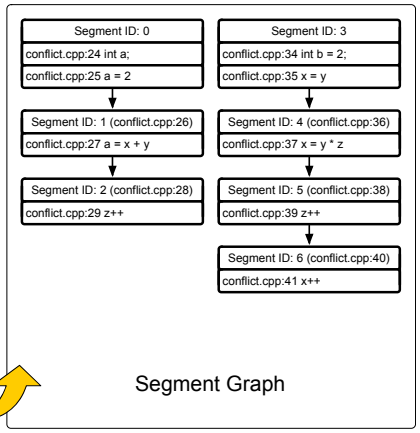
Conflict	Seg 1	Seg 2	Seg 3
Seg 1	True		
Seg 2		True	True
Seg 3		True	

Dedicated SystemC Compiler

- Segment Conflict Analysis:
  - Variable access analysis for Read, Write, and Read/Write
  - Example:

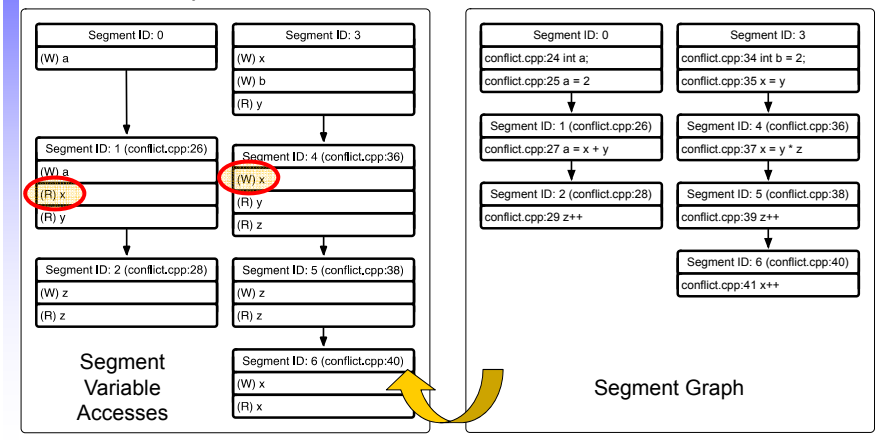
```
class Conflict: public sc_module {
  SC_CTOR(Conflict)
  { SC_THREAD(thread1);
    SC_THREAD(thread2);
  }
  int x, y, z;

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

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



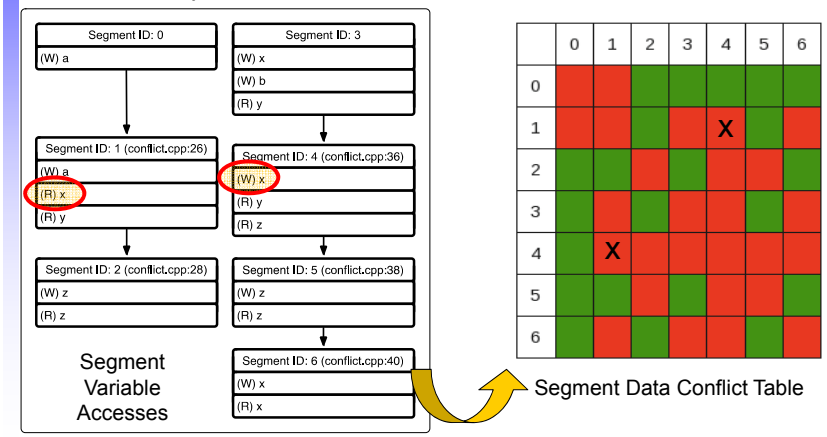
Dedicated SystemC Compiler

- Segment Conflict Analysis:
  - Variable access analysis for Read, Write, and Read/Write
  - Example:



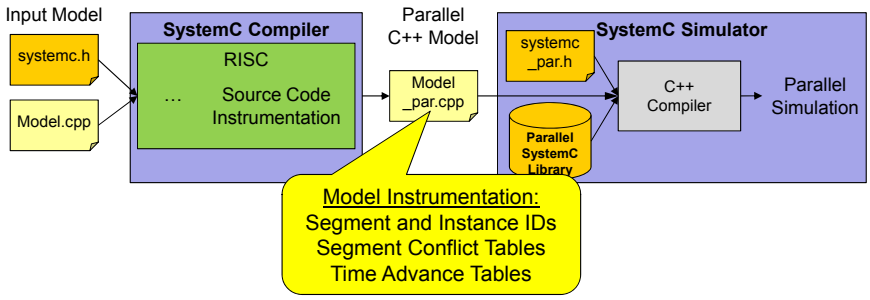
Dedicated SystemC Compiler

- Segment Conflict Analysis:
  - Variable access analysis for Read, Write, and Read/Write
  - Example:



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

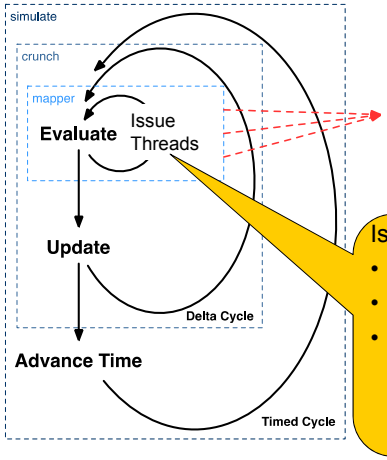


SystemC Compiler and Simulator

- 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*
  1) Segment and instance IDs
     - Threads identified by creator instance and current code location
  2) Data and event conflict tables
     - Segment concurrency hazards identified by fast table lookup (filtered for scope, instance path, references and port mapping)
  3) Current and next time advance tables
     - Prediction of future thread states
     - ➢ better scheduling decisions by looking ahead in time (future optimization)

Parallel SystemC Simulator

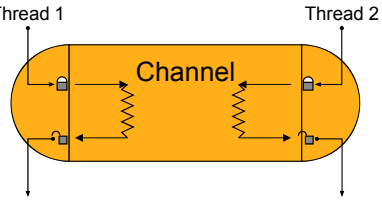
- Simulator kernel with Out-of-Order Parallel Scheduler
  - Conceptual OoO PDES execution



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

Parallel SystemC Simulator

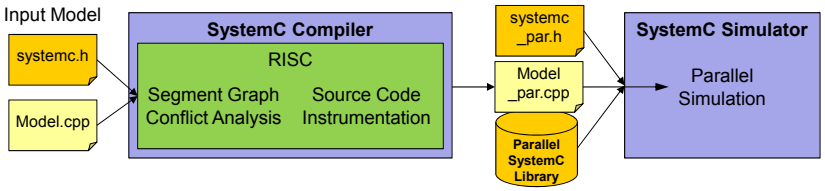
- Protection of Inter-Thread Communication
  - Need to comply with SystemC LRM [IEEE Std 1666™]
    - Cooperative (or co-routine) multitasking semantics
      - Threads can assume execution *"without interruption"*
  - Must protect inter-thread communication in channels!
    - Primitive SystemC channels
      - ➢ Static protection (special parallel SystemC headers, library)
    - User-defined hierarchical channels
      - ➢ Dynamic protection through source code instrumentation



Out-of-Order Parallel SystemC, UNSW, 2 March 2016 (c) 2016 R. Doemer, CECS 33

Demo and Experimental Results

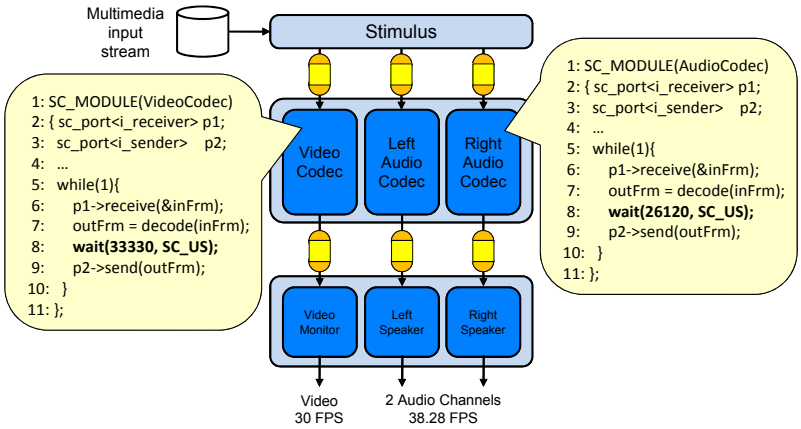
- Interactive Demonstration
  - Two Application Examples
    - DVD player (conceptual)
    - Mandelbrot renderer (embarrassingly parallel)
  - Compilation
    - Static analysis based on segment graph
    - Conflict analysis and source code instrumentation
  - Simulation
    - Accellera reference library (Posix-based, sequential)
    - RISC simulator library (Posix-based, out-of-order parallel)



Out-of-Order Parallel SystemC, UNSW, 2 March 2016 (c) 2016 R. Doemer, CECS 34

Example Model 1: DVD Player

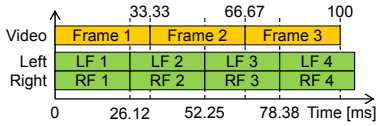
- DVD Player Example (conceptual)
  - Parallel video and audio decoding with different frame rates

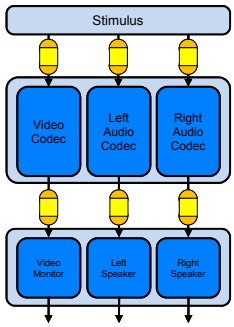


Example Model 1: DVD Player

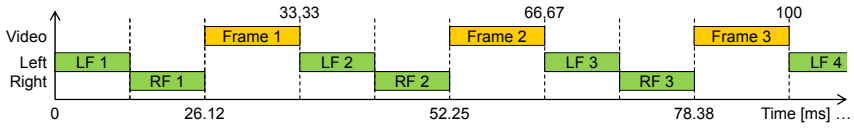
- DVD Player Example (conceptual)
  - Parallel video and audio decoding with different frame rates

1. Real time schedule: fully parallel



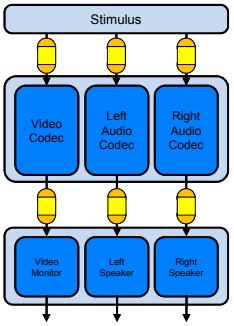


2. Reference simulator schedule (DES)

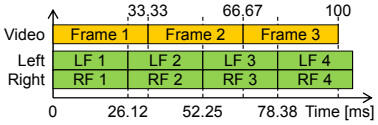


Example Model 1: DVD Player

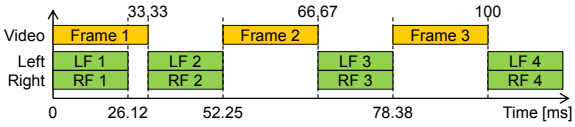
- DVD Player Example (conceptual)
  - Parallel video and audio decoding with different frame rates



1. Real time schedule: fully parallel

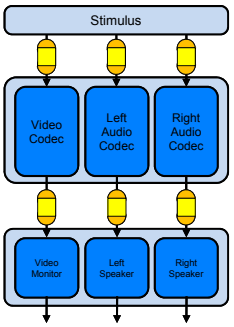


3. Synchronous parallel schedule (PDES)

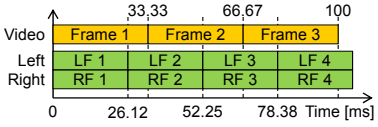


Example Model 1: DVD Player

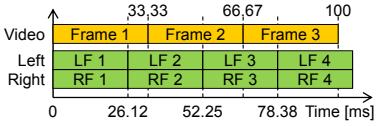
- DVD Player Example (conceptual)
  - Parallel video and audio decoding with different frame rates



1. Real time schedule: fully parallel

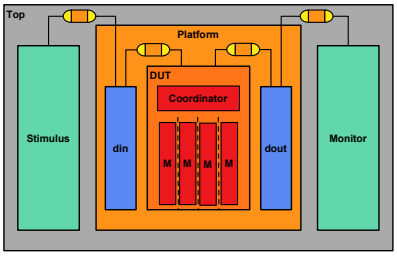


4. Out-of-order parallel schedule (OoO PDES)



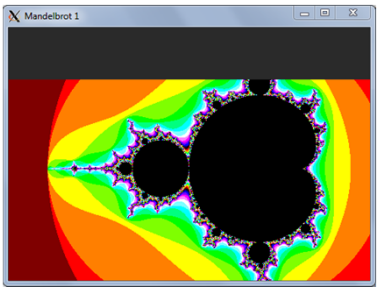
Example Model 2: Mandelbrot

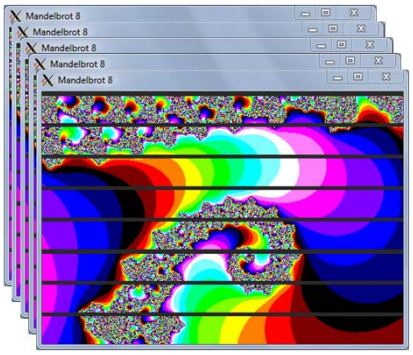
- Mandelbrot Renderer (Graphics Pipeline Application)
  - Mandelbrot Set
    - Mathematical set of points in complex plane
      - Two-dimensional fractal shape
    - High computation load
      - Recursive/iterative function
    - Embarrassingly parallel
      - Parallelism at pixel level
  - SystemC Model
    - TLM abstraction
    - Parallel slices
    - Configurable
    - Executable



Example Model 2: Mandelbrot

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





Experimental Results

- DVD Player Example (conceptual)
  - ➢ Parallel video and audio decoding with different frame rates
  - – Simulator run times on Intel® Xeon® multi-core host ('delta') (1 E3-1240 CPU, 3.4 GHz, 4 cores, 2 way hyper-threaded)
  - – RISC V0.2.1, Posix-thread based comparison

		Seq	Par	OoO
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

Experimental Results

- Mandelbrot Renderer Example
  - ➢ Graphics Pipeline Application, embarrassingly parallel
  - – Simulator run times on Intel® Xeon® multi-core host ('phi') (2 E5-2680 CPUs, 2.7 GHz, 8 cores, 2 way hyper-threaded)
  - – RISC V0.2.1, Posix-thread based comparison

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

Experimental Results

- Mandelbrot Renderer Example
  - ➢ Graphics Pipeline Application, embarrassingly parallel
  - – Simulator run times on Intel® Many Integrated Core (MIC) Architecture
  - ➢ Intel® Xeon Phi™ coprocessor
    - 5110P CPU at 1.052 GHz
    - 60 cores, 4 way hyper-threaded
    - Bidirectional ring interconnect, L2 chache
    - Appears as regular Linux machine with 240 cores!
  - ➢ Experimental result:
    - Traditional DES vs. synchronous PDES (no conflicts):
    - Run time (seq/par): 226.57 sec / 5.50 sec
    - ➢ **41x speedup**!

RISC Compiler and Simulator

- ➢ Out-of-Order Parallel SystemC Compiler and Simulator
- Open Source Prototype Implementation
  - – Alpha Release V0.2.1 published October 30, 2015
    - `http://www.cecs.uci.edu/~doemer/risc.html`
  - ➢ Source tar ball: `risc_v0.2.1.tar.gz`
  - ➢ Installation: `INSTALL, Makefile`
  - ➢ Doxygen documentation: `RISC API`
  - ➢ Doxygen documentation: `OoO Parallel SystemC API`
  - ➢ BSD license terms: `LICENSE`
- Downloads and feedback welcome!
  - ➢ Code hardening
  - ➢ Extension of supported parallel SystemC subset
  - ➢ Standardization…

Concluding Remarks

- Project on Advanced Parallel SystemC Simulation
  - Out-of-Order PDES on many-core host platforms
  - Maximum compliance with current execution semantics
- SystemC Compiler Integrated with Parallel Simulator
  - Segment Graph based static analysis for parallel execution
  - Model instrumentation and protection of communication
  - Out-of-order parallel scheduler, many-core platform support
- Open Source
  - RISC V0.2.1, working prototype implementation
  - Available at **www.cecs.uci.edu/~doemer/risc.html**
- Ongoing and Future Work
  - Code hardening and virtual platform integration (i.e. Simics®)
  - Collaboration with Accellera SystemC Language WG

References

- [CECS-TR-15-02] G. Liu, T. Schmidt, R. Dömer: *"RISC Compiler and Simulator, Alpha Release V0.2.1: Out-of-Order Parallel Simulatable SystemC Subset"*, Center for Embedded and Cyber-physical Systems, CECS, April 2015.
- [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.
- [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. (**Best Paper Award!**)
- [DATE'13] W. Chen, R. Dömer: *"Optimized Out-of-Order Parallel Discrete Event Simulation Using Predictions"*, Proceedings of DATE, Grenoble, France, March 2013.
- [IEEE D&T'13] W. Chen, X. Han, C. Chang, R. Dömer: *"Advances in Parallel Discrete Event Simulation for Electronic System-Level Design"*, IEEE Design & Test of Computers, vol. 30, no. 1, pp. 45-54, Jan.-Feb. 2013.
- [DATE'12] W. Chen, X. Han, R. Dömer: *"Out-of-Order Parallel Simulation for ESL Design"*, Proceedings of DATE, Dresden, Germany, March 2012.
- [ASPDAC'12] W. Chen, R. Dömer: *"An Optimizing Compiler for Out-of-Order Parallel ESL Simulation Exploiting Instance Isolation"*, Proceedings of ASPDAC, Sydney, Australia, February 2012.