Towards Parallel Simulation of Multi-Domain System Models

Keynote
DAC 2015 Workshop
System-to-Silicon Performance Modeling and Analysis

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

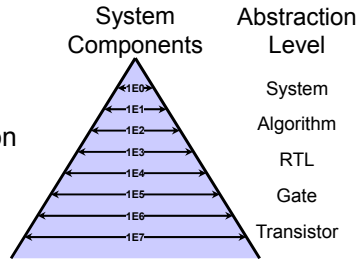
UCIrvine
University of California, Irvine

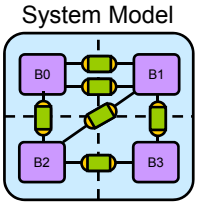
Towards Parallel Simulation…

- System Modeling
  - Models
  - Domains
  - Dimensions
- System Simulation
  - Discrete Event Simulation (DES)
  - Parallel Discrete Event Simulation (PDES)
  - Out-of-Order Parallel Discrete Event Simulation (OoO PDES)
- Parallel SystemC Project
  - Dependency Analysis using Segment Graphs
  - Early Experimental Results
- Concluding Remarks

System Modeling

- What is a Model?
  - A model is an abstraction of reality.
- What is Abstraction?
  - Abstraction is the intentional omission or simplification of details.
    - Reduces system complexity!
    - Allows to focus on the essentials!
- What is Modeling?
  - Creating and shaping a model:
    - Choosing and including properties of interest
    - Simplifying of characteristics of limited interest
    - Omitting of aspects of no interest
- Our Interest
  - Cyber-Physical and Embedded Systems





Towards Parallel Simulation of Multi-Domain System Models (c) 2015 R. Doemer, CECS 3

System Modeling Domains

- Traditional Properties of Interest in System Design
  - **Functionality** (execution)
    - Computation of functional result (data values)
    - Validation of correctness, Boolean value
  - **Performance** (estimation of speed, meeting deadlines)
    - Execution time t [s], optimization goal
  - **Structure** (number and type of components, estimation of cost,)
    - Chip area [mm^2], optimization goal
- New Domains of Increasing Importance
  - **Energy consumption** (i.e. battery run-time)
    - Power P [W] = E [J] / t [s], optimization goal with threshold
  - **Thermal behavior** (need for cooling)
    - Temperature T [^oC], threshold value, $T < T_{critical}$
  - **Reliability, degradation, aging**
    - Mean time between failures MTBF [s], optimization goal
  - **…**

Towards Parallel Simulation of Multi-Domain System Models (c) 2015 R. Doemer, CECS 4

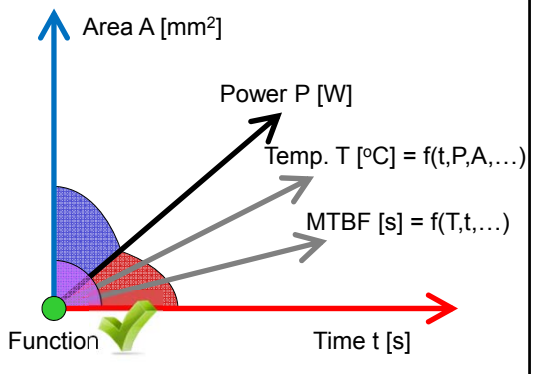
System Modeling Domains

- Traditional Properties of Interest in System Design
  - **Functionality** (execution)
    - Computation of functional result (data values)
    - Validation of correctness, Boolean value
  - **Performance** (estimation
    - Execution time t [s], optim
  - **Structure** (number and typ
    - Chip area [mm^2], optimiza
- New Domains of Increasing
  - **Energy consumption** (i.e. battery run-time)
    - Power P [W] = E [J] / t [s], optimization goal with threshold
  - **Thermal behavior** (need for cooling)
    - Temperature T [^{o}C], threshold value, T < $T_{critical}$
  - **Reliability, degradation, aging**
    - Mean time between failures MTBF [s], optimization goal
  - …

Extra-Functional Properties
(e.g. [Ramesh'12], [Hartmann'15])

System Modeling Dimensions

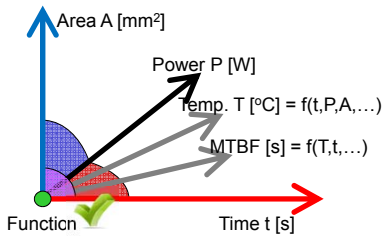
- Traditional Properties
  - Functionality (check!)
  - Performance (max!)
  - Cost (min!)
- New Properties
  - Power (min!)
  - Thermal effects (min!)
  - Reliability (max!)
  - Degradation, Aging
  - …



- *Properties in different dimensions can be modeled independently!*
- *Opportunity to exploit dimensions for parallel simulation!*

System Modeling Dimensions

- SystemC Example:
  - Function
  - Structure
  - Timing
  - Power consumption
  - Thermal behavior
  - Reliability degradation




```
SC_MODULE(Component)
{ sc_port<i_data> DataIn, DataOut;
  SC_CTOR(Component)
  { SC_THREAD(main);
  }
  void main()
  { ...
    x = f(DataIn->read());
    wait(delay1(), SC_NS);
    consume(power1(), mW);
    dissipate(temp1(), degC);
    age(mtbf1(), SC_SEC);
    ...
    while(cond(x))
    { ...
      y = g(x);
      wait(delay2(), SC_NS);
      consume(power2(), mW);
      dissipate(temp2(), degC);
      age(mtbf2(), SC_SEC);
    }
    DataOut->write(y);
  }
};
```

Towards Parallel Simulation of Multi-Domain System Models (c) 2015 R. Doemer, CECS 7

System Simulation

- Traditional Discrete Event Simulation
  - well-suited for functional validation and performance vs. cost trade-offs
  - but cannot cope with the additional complexities and extra-functional properties of new domains
    - ➢ Growing complexity impacts execution speed!
- ➢ True system simulation must advance to
  1) efficiently integrate new domains and dimensions
  2) fully exploit parallel execution
     - ➢ Utilize parallelism to ensure scalability!
- Example project:
  - Out-of-Order Parallel Simulation of SystemC Models
  - ➢ Parallel, fast, and accurate

Towards Parallel Simulation of Multi-Domain System Models (c) 2015 R. Doemer, CECS 8

Parallel Simulation, Key Points

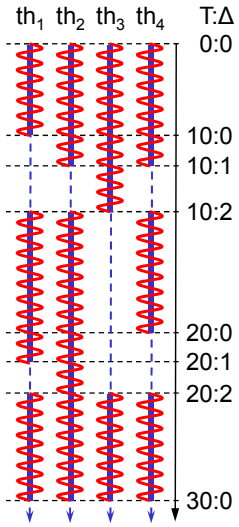
- Project on Advanced Parallel SystemC Simulation
  - Out-of-Order PDES on many-core host platforms
  - Maximum compliance with current execution semantics
  - Supported with funding by Intel® Corp.
- Introduction of a Dedicated SystemC Compiler
  - Recoding Infrastructure for SystemC (RISC)
  - Advanced 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

Discrete Event Simulation

- 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] 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*!

Discrete Event Simulation (DES)

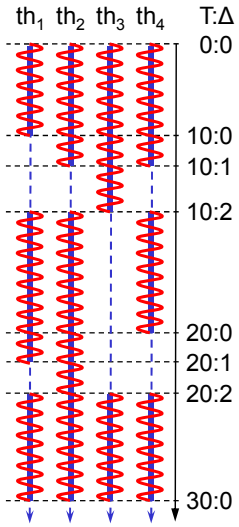
- Traditional DES
  - Concurrent threads of execution
  - Managed by a central scheduler
  - Driven by events and time advances
    - Delta-cycle
    - Time-cycle
  - ➢ Partial temporal order with barriers
- IEEE Standard Simulator
  - SystemC reference simulator uses cooperative multi-threading
  -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



Towards Parallel Simulation of Multi-Domain System Models (c) 2015 R. Doemer, CECS 11

Parallel Discrete Event Simulation (PDES)

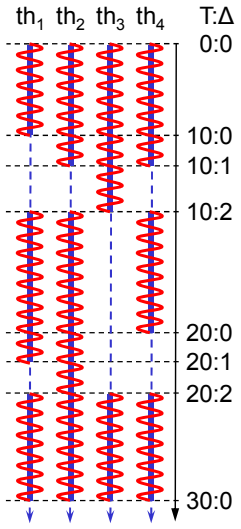
- Parallel DES
  - Threads execute in parallel *iff*
    - in the same delta cycle, *and*
    - in the same time cycle
  - ➢ Significant speed up!
  - *Synchronous* PDES: Cycle boundaries are *absolute barriers*!
- Aggressive Parallel DES
  - Conservative Approaches
    - Careful static analysis prevents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



Towards Parallel Simulation of Multi-Domain System Models (c) 2015 R. Doemer, CECS 12

Out-of-Order Parallel DES

- 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!
  - Can utilize advanced compiler for static data conflict analysis
  - ➢ Allows as many threads in parallel as possible
  - ➢ Significantly higher speedup!
    - Results at [DATE'12], [IEEE TCAD14]
  - ➢ Fully preserves…
    - ➢ DES execution semantics
    - ➢ Accuracy in results and timing



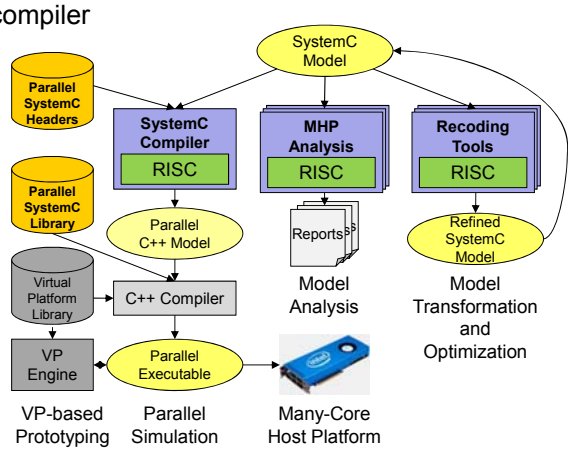
Out-of-Order PDES Technology

- 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

Project Overview and Tool Flow

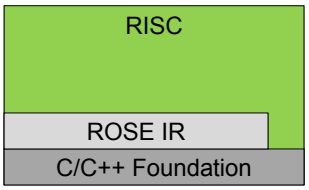
- Research and Development Tasks

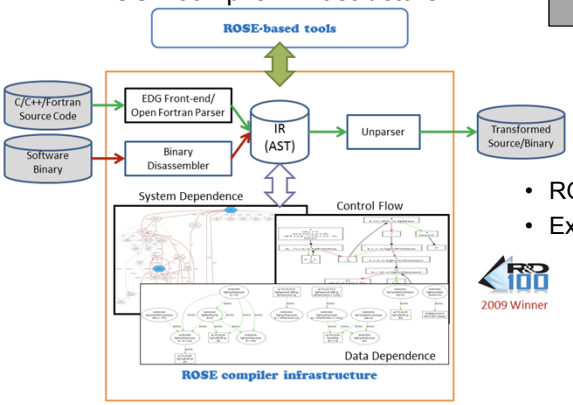
1) Dedicated SystemC compiler (RISC infrastructure)
2) Parallel SystemC headers and library
3) Performance tuning for many-core hosts
4) Virtual Platform (VP) integration
5) Model analysis (may-happen-in-parallel, MHP)
6) Model recoding, transformation and optimization



R&D Task 1: SystemC Compiler

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



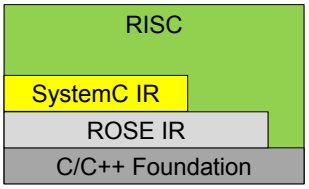


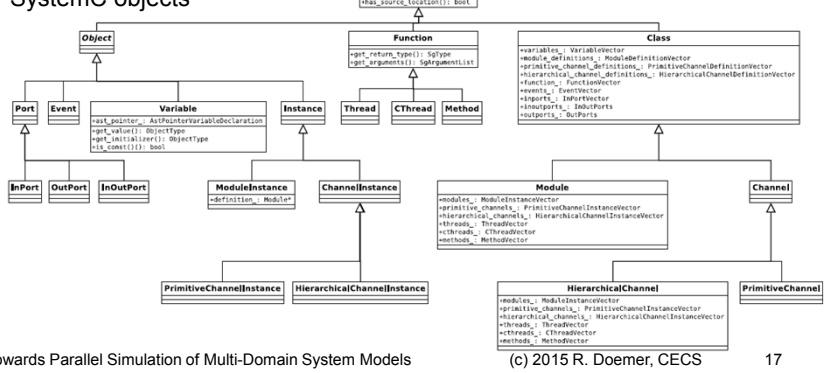
- ROSE Internal Representation
- Explicit support for
  - Source code analysis
  - Source-to-source transformations

2009 Winner

R&D Task 1: SystemC Compiler

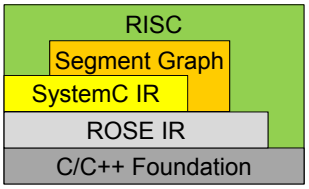
- RISC Software Stack
  - ➢ *Recoding Infrastructure for SystemC*
  - – SystemC Internal Representation
- Class hierarchy to represent SystemC objects

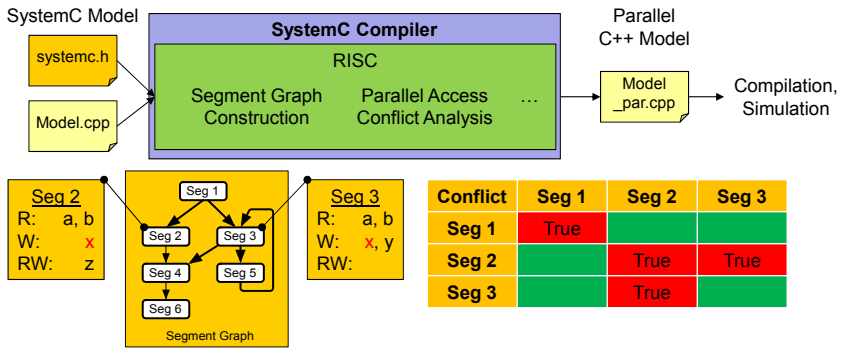




R&D Task 1: SystemC Compiler

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

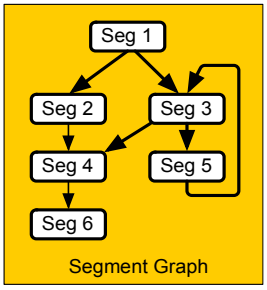




Conflict	Seg 1	Seg 2	Seg 3
Seg 1	True		
Seg 2		True	True
Seg 3		True	

R&D Task 1: SystemC Compiler

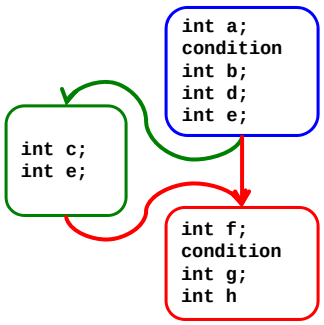
- Segment Graph Construction
  - *Segment Graph* is a directed graph
    - Nodes: *Segments*
    - ➢ Code statements executed between two scheduling steps
      - Expression statements
      - Control flow statements (if, while, …)
      - Function calls
    - Edges: *Segment boundaries*
    - ➢ Primitives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



R&D Task 1: SystemC Compiler

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

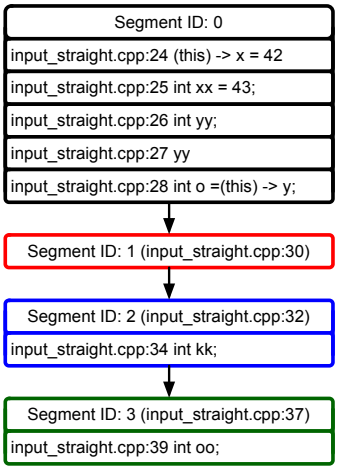
```
int a;
if(cond) {
    int b;
    wait(1);
    int c;
} else {
    int d;
}
int e;
wait(2);
int f;
while(cond) {
    int g;
}
int h;
```



R&D Task 1: SystemC Compiler

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

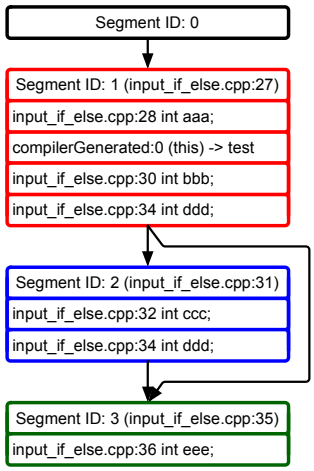
```
void straight()
{
     x = 42;
     int xx = 43;
     int yy;
     yy;
     int o = y;

     wait(10, SC_NS);

     wait();

     int kk;

     wait();

     int oo;
}
```



R&D Task 1: SystemC Compiler

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

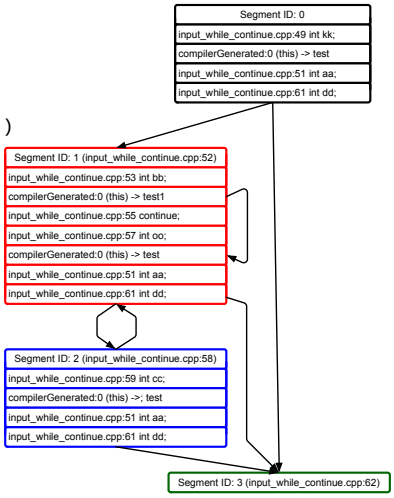
```
void if_statement()
{
     wait();
     int aaa;
     if(test) {
         int bbb;
         wait();
         int ccc;
     }
     int ddd;
     wait();
     int eee;
}
```



R&D Task 1: SystemC Compiler

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

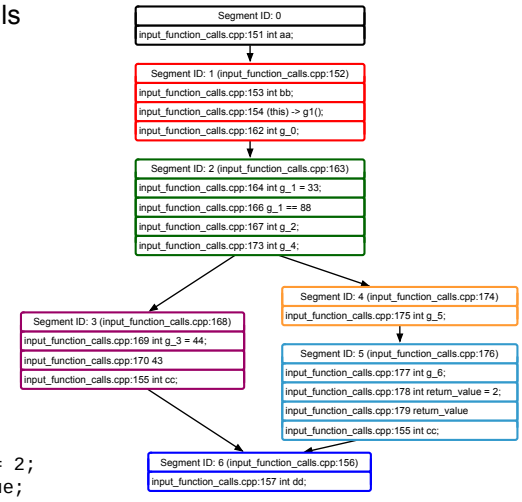
```
void while_continue_statement()
{
     int kk;
     while(test){
         int aa;
         wait();
         int bb;
         if(test1) {
            continue;
         }
         int oo;
         wait();
         int cc;
     }
     int dd;
     wait();
}
```



R&D Task 1: SystemC Compiler

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

```
void f()    int g1()
{           {
  int aa;     int g_0;
  wait();     wait();
  int bb;     int g_1 = 33;
  g1();       if(g_1 == 88) {
  int cc;        int g_2;
  wait();        wait();
  int dd;        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;
               }
```

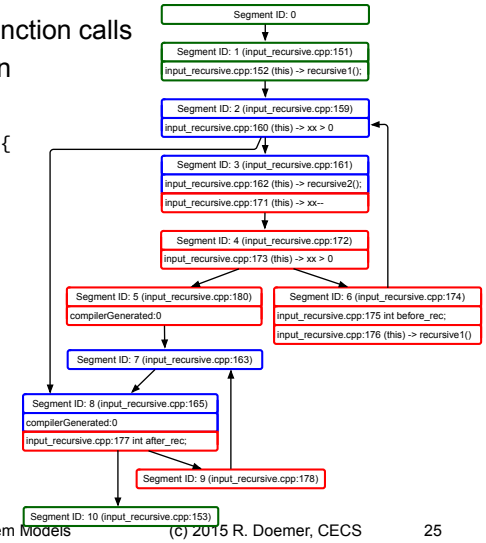


R&D Task 1: SystemC Compiler

- Segment Graph Construction: Current Status
  - 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;
}
```



R&D Task 1: SystemC Compiler

- Segment Conflict Analysis
  - Need to comply with SystemC LRM [IEEE Std 1666™]
    - Cooperative (or co-routine) multitasking semantics
      - *"process instances execute without interruption"*
      - System designer *"can assume that a method process will execute in its entirety without interruption"*
      - ➢ A parallel implementation *"would be obliged to analyze any dependencies between processes and constrain their execution to match the co-routine semantics."*
  - Must avoid race conditions when using shared variables!
    - ➢ Prevent conflicting segments to be scheduled in parallel

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

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

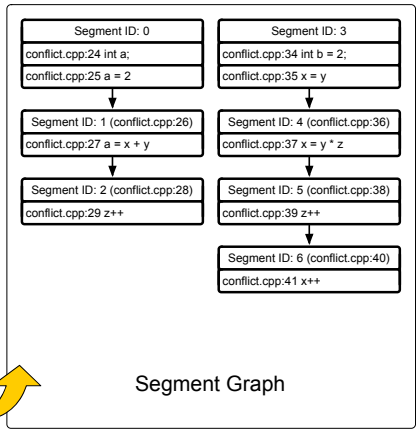
Conflict	Seg 1	Seg 2	Seg 3
Seg 1	True		
Seg 2		True	True
Seg 3		True	

R&D Task 1: SystemC Compiler

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

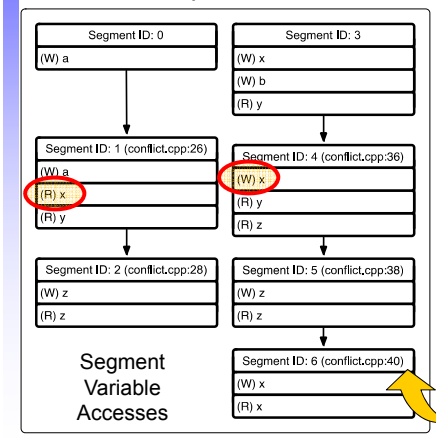
```
class Conflict: public sc_module {
  SC_CTOR(Conflict)
  { SC_THREAD(thread1);
    SC_THREAD(thread2);
  }
  int x, y, z;

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

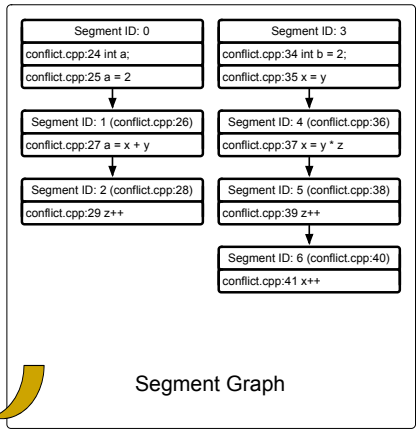
```
void thread2()
{
  int b = 2;
  x = y;
  wait();
  x = y * z;
  wait();
  z++;
  wait();
  x++;
}
```



R&D Task 1: SystemC Compiler

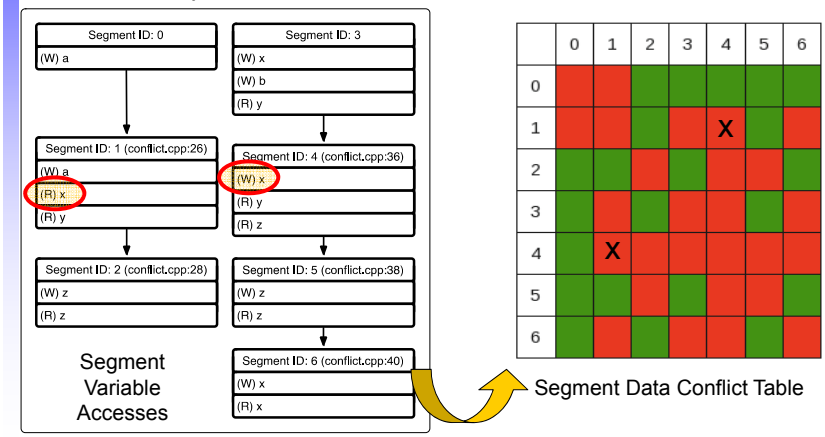
- Segment Conflict Analysis: Current Status
  - Variable access analysis for Read, Write, and Read/Write
  - Example:





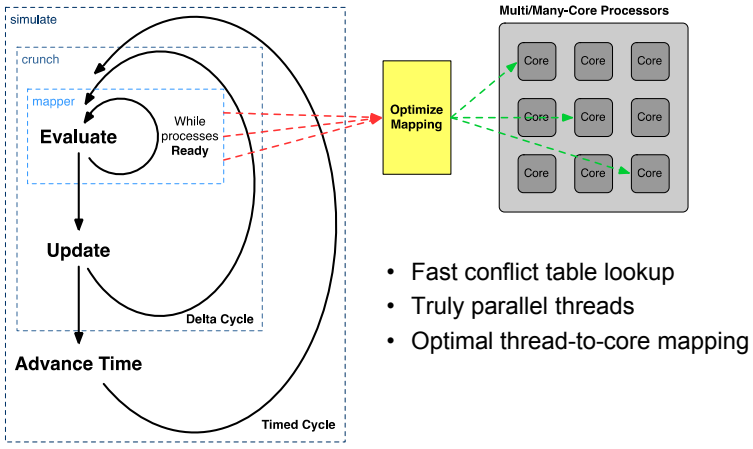
R&D Task 1: SystemC Compiler

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



R&D Task 2: Parallel SystemC Library

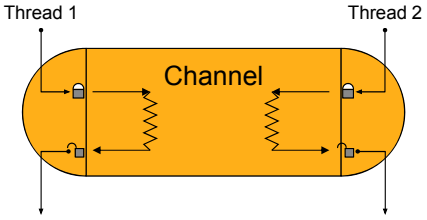
- Parallel Simulator with Out-of-Order Scheduler
  - OoO PDES execution



- Fast conflict table lookup
- Truly parallel threads
- Optimal thread-to-core mapping

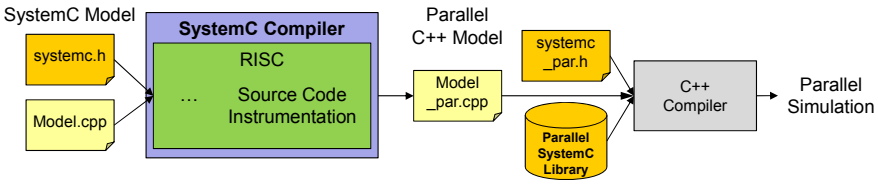
R&D Task 2: Parallel SystemC Library

- Inter-Thread Communication Protection
  - Need to comply with SystemC LRM [IEEE Std 1666™]
    - Cooperative (or co-routine) multitasking semantics
      - Execution *"without interruption"*
  - Must protect inter-thread communication in channels!
    - ➢ Insert a mutex lock into channel instances
      - To lock the channel on thread entry
      - To unlock the channel on thread exit



R&D Task 2: Parallel SystemC Library

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



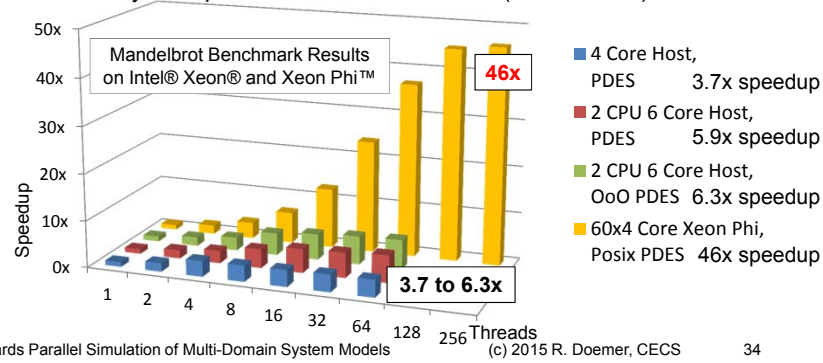
Early Experimental Results

- Intel® Many Integrated Core Architecture
- ➢ Intel® Xeon Phi™ Coprocessor
  - Provides
    - 60 processor cores
    - 4 hyper-threads per core
    - ➢ 240 parallel hardware threads!
  - Hardware Features
    - Vector processing unit (VPU)
    - Extended Math Unit (EMU) for transcendental operations
    - Bidirectional ring interconnect
  - Peak performance
    - ➢ over 1 teraFLOPS (double-precision)
  - ➢ Uses familiar and standard programming models
    - ➢ Appears as a regular Linux machine with 240 cores!

Towards Parallel Simulation of Multi-Domain System Models (c) 2015 R. Doemer, CECS 33

Early Experimental Results

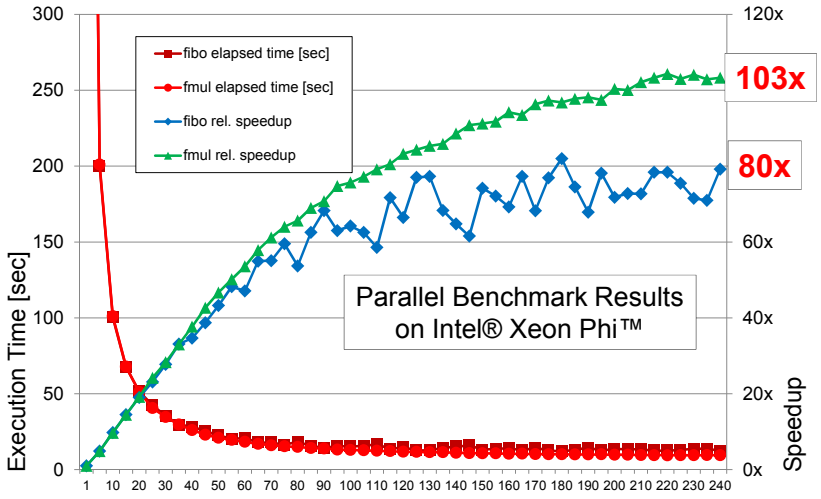
- Graphics Application: Mandelbrot Set Renderer
  - Experimental Results
    - Sequence of 100 Mandelbrot images (640x448, depth 4096)
    - Manually created PDES model (Posix-threads based)
    - Multi-core platforms: *Intel® Xeon® CPUs* (4 cores, 2x6 cores)
    - Many-core platform: *Intel® Xeon Phi™* (60 x 4 cores)



Towards Parallel Simulation of Multi-Domain System Models (c) 2015 R. Doemer, CECS 34

Early Experimental Results

- Experimental Results (Intel® Xeon Phi Coprocessor, 60x4 cores)



Towards Parallel Simulation of Multi-Domain System Models (c) 2015 R. Doemer, CECS 35

Concluding Remarks

- Project on Advanced Parallel SystemC Simulation
  - Out-of-Order PDES on many-core host platforms
  - Maximum compliance with current execution semantics
  - Supported with funding by Intel® Corp.
- Introduction of a Dedicated SystemC Compiler
  - Recoding Infrastructure for SystemC (RISC)
  - Advanced static analysis for parallel execution
  - Model instrumentation, code optimization, transformation
- 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

Towards Parallel Simulation of Multi-Domain System Models (c) 2015 R. Doemer, CECS 36

Conclusion

- System simulation must

1) integrate new domains for extra-functional properties
   - Performance
   - Structure
   - Energy consumption
   - Thermal behavior
   - Reliability, degradation, aging
2) exploit parallel execution to ensure scalability

- Independent domains form new dimensions
  - ➢ Exploit independent dimensions for parallel simulation

References

- [Hartmann'15] P. Hartmann, K. Gruettner, W. Nebel: *"Advanced SystemC Tracing and Analysis Framework for Extra-Functional Properties"*, Proceedings of ARC Symposium, Bochum, Germany, 2015.
- [UTokyo'15] R. Dömer, G. Liu, T. Schmidt: *"Out-of-Order Parallel Simulation of SystemC Models on Many-Core Architectures"*, Presentation at University of Tokyo, Japan, January 2015.
- [IEEE TCAD14] 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.
- [Ramesh'12] U. Ramesh: *"A taxonomy for extra-functional properties of embedded system"*, Master's thesis, Maelardalen University, Sweden, 2012.
- [DATE'12] W. Chen, X. Han, R. Dömer: *"Out-of-Order Parallel Simulation for ESL Design"*, Proceedings of DATE, Dresden, Germany, March 2012.
- [Maehne'09] T. Maehne, A. Vachoux: *"Supporting Dimensional Analysis in SystemC-AMS"*, Proceedings of BMAS Workshop, San Jose, 2009.