


Advances in Parallel Simulation of System Models


Rainer Dömer
doemer@uci.edu

With contributions by Weiwei Chen and Xu Han

Center for Embedded Computer Systems
University of California, Irvine




UCIrvine
University of California, Irvine




This work has been supported in part by NSF Award #0747523.

Embedded Systems


- System embedded into another system
 - Constraints from external input
 - Application specific
- Omnipresent in our environment
 - In many application domains
 - In 2005 [Source Netrino]
 - Only 2% of all processors in workstations
 - Remaining 8.8 billion in embedded systems
 - Pervasive




Source: P. Chou, UCI




Source: Edumaticator




Source: Miele
EECS Colloquium, October 17, 2012



Source: Philips



Source: www.trouper.com

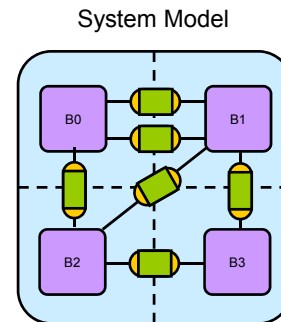


Source: www.medicacorp.com/

(c) 2012 R. Doemer, et.al. 2

Embedded System Design

- System Level Modeling
 - 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
 - Channels and Interfaces
 - Behaviors / Modules



EECS Colloquium, October 17, 2012

(c) 2012 R. Doemer, et.al.

3

Outline

- Embedded System Validation
- Parallel Simulation
 - Traditional Discrete Event Simulation (DES)
 - Parallel Discrete Event Simulation (PDES)
 - Potential and Reality: Experimental Results
 - Parallel Benchmarks
 - Embedded Application Examples
- Advanced Parallel Simulation
 - Out-of-Order Parallel DES
 - Approach
 - Experiments and Results
- Conclusions

EECS Colloquium, October 17, 2012

(c) 2012 R. Doemer, et.al.

4

Embedded System Validation

- 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 { }`
 - Symmetric Multi-Processor (SMP) architecture
 - Multi-core host PCs readily available
 - Many-core machines are coming

EECS Colloquium, October 17, 2012 (c) 2012 R. Doemer, et.al. 5

Related Work: Faster Simulation

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

Hardware-based Acceleration

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

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. [IEEEED&T'11]
- Yun et al. [TCAD'12]

EECS Colloquium, October 17, 2012 (c) 2012 R. Doemer, et.al. 6

Discrete Event Simulation

- Traditional Discrete Event (DE) Simulation
 - Execution semantics used in
 - SLDLs, i.e. SpecC, SystemC
 - HDLs, i.e. VHDL, Verilog
 - Non-deterministic sequential execution of “parallel” threads
 - *Delta cycles*
- 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 \emptyset)
 - **WAITFOR**: set of threads suspended by `waitfor` (initially \emptyset)
 - 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 \emptyset

EECS Colloquium, October 17, 2012 (c) 2012 R. Doemer, et.al. 7

Discrete Event Simulation

- Simulation Algorithm

```

graph TD
    Start([Start]) --> Select[Select thread t ∈ READY, execute t]
    Select --> Notify{notify}
    Notify -- YES --> AddN[Add notified events to N]
    AddN --> Wait{wait}
    Wait -- YES --> MoveWait[Move t ∈ READY to WAIT]
    MoveWait --> Waitfor{waitfor}
    Waitfor -- YES --> MoveWaitfor[Move t ∈ READY to WAITFOR]
    Waitfor -- NO --> ReadyEmpty1{READY = ∅}
    MoveWaitfor --> ReadyEmpty1
    MoveWait --> ReadyEmpty1
    AddN --> ReadyEmpty1
    ReadyEmpty1 -- YES --> MoveWaitReady[Move all t ∈ WAIT waiting for events e ∈ N to READY]
    MoveWaitReady --> SetN[Set N = ∅]
    SetN --> ReadyEmpty2{READY = ∅}
    ReadyEmpty2 -- NO --> UpdateTime[Update simulation time, move earliest t ∈ WAITFOR to READY]
    UpdateTime --> ReadyEmpty3{READY = ∅}
    ReadyEmpty3 -- YES --> Stop([Stop])
    ReadyEmpty3 -- NO --> Select
    
```

EECS Colloquium, October 17, 2012 (c) 2012 R. Doemer, et.al. 8

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 TLM Simulators
 - Both SystemC and SpecC use cooperative multi-threading
 - A single thread is active at any time!
 - Cannot exploit multiple parallel cores
 - Example: SystemC

EECS Colloquium, October 17, 2012 (c) 2012 R. Doemer, et.al. 9

Preemptive Discrete Event Simulation

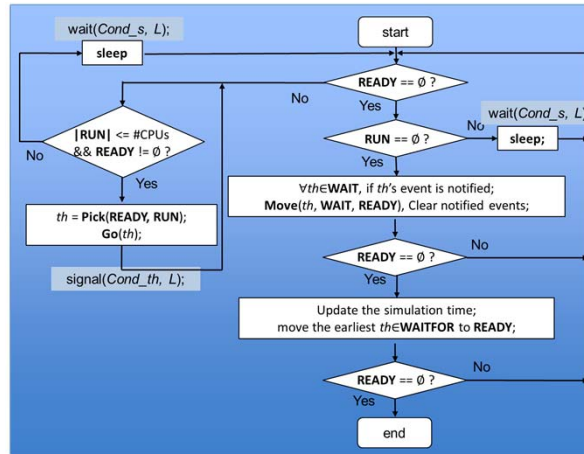
- SLDL Execution Semantics
 - SystemC prescribes *Cooperative Multi-Threading*
 - SystemC LRM defines: *"process instances execute without interruption"*
 - Preemptive scheduling forbidden!
 - SpecC specifies *Preemptive Multi-Threading*
 - SpecC LRM defines: *"preemptive execution", "No atomicity is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

EECS Colloquium, October 17, 2012 (c) 2012 R. Doemer, et.al. 10

Parallel Discrete Event Simulation (PDES)

- Parallel DE Simulation Algorithm

- Threads managed in READY queue
- Scheduler picks N threads and executes them in parallel
- N = number of available CPU cores
- Time advances
 - In delta-cycle
 - In timed-cycle



Parallel Discrete Event Simulation (PDES)

- Synchronization is required!
 - Need to protect shared data structures by *locks* for mutual exclusive access by concurrent threads
- 1. Protecting scheduling resources
 - Central lock for scheduler
 - Condition variable for each thread
- 2. Protecting communication
 - One lock per channel instance
 - Lock protects critical region
 - Channel acts as *monitor* for encapsulated variables
 - Locks and locking methods are automatically inserted!

```

1 send(d)
  {
3   Lock(this->L);
   while(n >= size){
5     ws ++;
     wait(eSend);
7     ws --;
   }
9   buffer.store(d);
   if(wr){
11    notify(eRecv);
   }
13  unLock(this->L);
  }
    
```

Parallel Discrete Event Simulation (PDES)

- Life-cycle of a Thread in our Multi-core Parallel Simulator
- Locks and condition variables guarantee safe synchronization and communication of concurrent threads

EECS Colloquium, October 17, 2012 (c) 2012 R. Doemer, et.al. 13

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!
- Naïve Promise: Linear Speedup with SMP!
 - But: Amdahl's Law still applies!
- Reality check: Experiments to evaluate
 - Potential vs. real applications

EECS Colloquium, October 17, 2012 (c) 2012 R. Doemer, et.al. 14

PDES Experiments and Results

- What is the Potential Amount of Parallelism that we can exploit?
 - Experimental Setup:
 - Parallel SpecC Simulator on SMP Host PC
 - 2 Intel Xeon X5650 CPUs at 2.66 GHz
 - 6 cores each
 - 2 hyper-threads per core
 - 24 parallel processing units available!
 - Experiments:
 - 2 highly parallel benchmarks
 - Upper bound for SMP capabilities of host PC
 - 2 typical embedded applications
 - Realistic industrial use cases

EECS Colloquium, October 17, 2012

(c) 2012 R. Doemer, et.al.

15

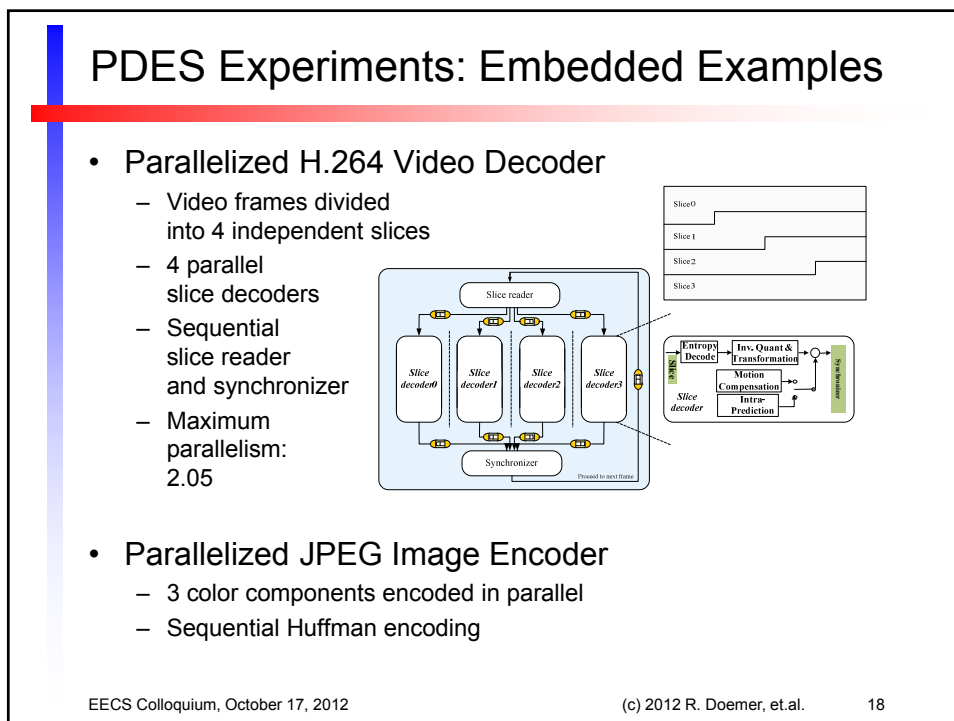
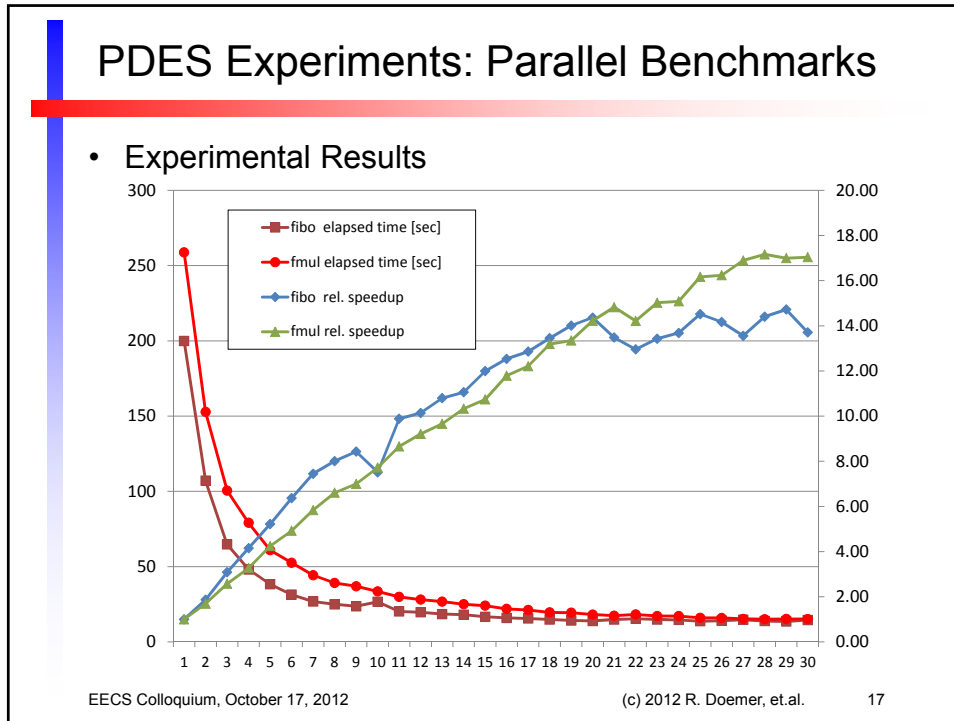
PDES Experiments: Parallel Benchmarks

- Parallel Floating-point Multiplications (**fmul**)
 - 10 million floating-point multiplications
 - 256 parallel instances
 - No communication, no shared variables
 - Balanced load
 - evenly distributed
- Parallel Fibonacci Calculation (**fibonacci**)
 - Fibonacci-series calculation
 - $fib(n) = fib(n-1) + fib(n-2)$, where $fib(0) = 0$, $fib(1) = 1$
 - Up to 256 parallel instances
 - Parallel decomposition up to max. depth of 8, then classic recursive calculation
 - Shared variables for input, output (plus a few counters)
 - Imbalanced load
 - distributed by the nature of Fibonacci numbers
 - converges against 1.618 (*golden ratio*)

EECS Colloquium, October 17, 2012

(c) 2012 R. Doemer, et.al.

16



PDES Experiments: Embedded Examples

- Simulation Results for H.264 Video Decoder

Simulator Par. issued threads:		Reference	Multi-Core					
		n/a	1		2		4	
		sim. time	sim. time	speedup	sim. time	speedup	sim. time	speedup
models	spec	20.80s (99%)	21.12s (99%)	0.98	14.57s (146%)	1.43	11.96s (193%)	1.74
	arch	21.27s (97%)	21.50s (97%)	0.99	14.90s (142%)	1.43	12.05s (188%)	1.76
	sched	21.43s (97%)	21.72s (97%)	0.99	15.26s (141%)	1.40	12.98s (182%)	1.65
	net	21.37s (97%)	21.49s (99%)	0.99	15.58s (138%)	1.37	13.04s (181%)	1.64
	tlm	21.64s (98%)	22.12s (98%)	0.98	16.06s (137%)	1.35	13.99s (175%)	1.55
comm		26.32s (96%)	26.25s (97%)	1.00	19.50s (133%)	1.35	25.57s (138%)	1.03
maximum speedup		1.00	1.00		1.52		2.05	

- Simulation Results for JPEG Image Encoder

Simulator Par. issued threads:		Reference	Multi-Core					
		n/a	1		2		4	
		sim. time	sim. time	speedup	sim. time	speedup	sim. time	speedup
models	spec	5.54s (99%)	5.97s (99%)	0.93	4.22s (135%)	1.31	3.12s (187%)	1.78
	arch	5.52s (99%)	6.07s (99%)	0.91	4.28s (135%)	1.29	3.15s (188%)	1.75
	sched	5.89s (99%)	6.38s (99%)	0.92	5.48s (108%)	1.07	5.47s (113%)	1.08
	net	11.56s (99%)	49.3s (99%)	0.23	40.63s (131%)	0.28	37.97s (128%)	0.30

➤ Multi-core parallelism significantly reduces simulation time!

PDES Experiments: Embedded Examples

- More Simulation Results for H.264 Video Decoder

Simulator Par. issued threads:		Reference	Multi-Core	
		n/a	24	
		sim. time	sim. time	speedup
models	spec	37.71s (90%)	21.01s (173%)	1.79
	arch	38.26s (90%)	21.26s (171%)	1.80
	sched	38.10s (91%)	22.31s (166%)	1.71
	net	38.18s (91%)	22.45s (166%)	1.70
	tlm	39.17s (90%)	22.32s (167%)	1.75
	comm	49.92s (90%)	33.55s (156%)	1.49

- More Simulation Results for JPEG Image Encoder

Simulator Par. issued threads:		Reference	Multi-Core	
		n/a	24	
		sim. time	sim. time	speedup
models	spec	3.46s (93%)	2.82s (146%)	1.23
	arch	3.92s (94%)	2.83s (147%)	1.39
	sched	4.64s (93%)	3.61s (137%)	1.29
	net	13.87s (97%)	45.96s (130%)	0.30

➤ Parallelism available in application is quite limited!

PDES Summary

- Embedded System Validation relies on Simulation
 - Discrete Event Simulation (DES)
 - Reference simulators execute sequentially, one thread at a time
- Parallel Discrete Event Simulation (PDES)
 - Exploits available parallelism in system models
 - Allows efficient execution on multi-core hosts
 - Shared data structures can be automatically protected
 - Channels define critical regions
 - Necessary synchronization can be automatically inserted
- Experimental Results
 - Multi-core PDES shows significant simulation speedup
 - Speedup is limited by explicitly specified parallelism!

EECS Colloquium, October 17, 2012 (c) 2012 R. Doemer, et.al. 21

Beyond Regular 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
- Aggressive Parallel DES
 - Conservative Approaches
 - Careful static analysis prevents conflicts
 - Optimistic Approaches
 - Conflicts are detected and addressed (*roll back*)

EECS Colloquium, October 17, 2012 (c) 2012 R. Doemer, et.al. 22

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!
 - Needs compiler support for static data conflict analysis!
 - Preserves the accuracy of event handling and simulation time
 - Allows as many threads in parallel as possible
 - Results in higher speedup!
 - Examples: [DATE'12, ASPDAC'12]

EECS Colloquium, October 17, 2012 (c) 2012 R. Doemer, et.al. 23

OoO PDES Simulation States

- Break global temporal barrier
 - localize simulation time ($T:\Delta$) to the threads.
 - attach timestamps to the events.

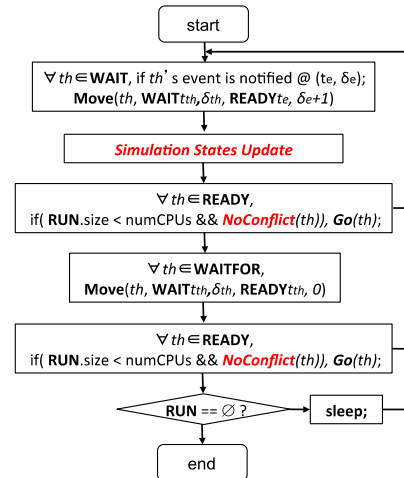
Static States of Regular PDES

Dynamic States of OoO PDES

EECS Colloquium, October 17, 2012 (c) 2012 R. Doemer, et.al. 24

OoO PDES Scheduling

- No need to wait till the global cycle boundaries.
- Dynamically update simulation states.
- Conflicts checking
 - **Data hazard:** RAW, WAR, WAW for shared variables.
 - **Conflicts Prediction:** check at runtime according to the simulation states.



EECS Colloquium, October 17, 2012

(c) 2012 R. Doemer, et al.

25

OoO PDES Conflict Analysis

- Compiler support for data conflict analysis
 - **Segment:**
SLDL code executed by a thread between two scheduling steps.
 - **Segment Boundary:**
SLDL statements which call the scheduler, e.g. wait, par, etc.
 - **Segment Graph:**
 - the connections among segments
 - derived from the control flow graph (CFG)
 - data access information for each segments
 - **Segment data conflict table:**
referred to at **runtime** for fast scheduling decision.

EECS Colloquium, October 17, 2012

(c) 2012 R. Doemer, et al.

26

OoO PDES Segment Graph

```

1: #include <stdio.h>
2: int array[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
3: behavior B(in int begin, in int end, out int sum)
4: { int i;
5: void main(){
6:   int tmp; tmp = 0; i = begin;
7:   waitfor 1; // v3, segment 3 starts
8:   while(i <= end){
9:     waitfor 2; // v4, segment 4 starts
10:    tmp += array[i]; i++;
11:   }
12:   sum = tmp; }
13: };
14: behavior Main()
15: {int sum1, sum2;
16:  B b1(0, 4, sum1); B b2(5, 9, sum2);
17:  int main(){
18:    par // v1, segment 1 starts
19:    b1.main();
20:    b2.main();
21:  } // v2, segment 2 starts
22:  printf("summation 1 is :%d \n", sum1);
23:  printf("summation 2 is :%d \n", sum2); }};
    
```

The graph shows a parallel structure. **thread_Main** starts with **seg0**, which branches into **thread_b1** and **thread_b2**. Both threads execute **seg1**, followed by a **waitfor 1** node. After **waitfor 1**, they execute **seg3**, followed by a **waitfor 2** node. Finally, they execute **seg4**. Both threads then merge at a **par end** node, which leads to **seg2** in **thread_Main**. The **waitfor 2** nodes in **thread_b1** and **thread_b2** are connected to **seg2**, indicating a dependency on the completion of **seg4** in both threads.

(c) 2012 R. Doemer, et al. 27

OoO PDES Segment Conflict Table

```

1: #include <stdio.h>
2: int array[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
3: behavior B(in int begin, in int end, out int sum)
4: { int i;
5: void main(){
6:   int tmp; tmp = 0; i = begin;
7:   waitfor 1; // v3, segment 3 starts
8:   while(i <= end){
9:     waitfor 2; // v4, segment 4 starts
10:    tmp += array[i]; i++;
11:   }
12:   sum = tmp; }
13: };
14: behavior Main()
15: {int sum1, sum2;
16:  B b1(0, 4, sum1); B b2(5, 9, sum2);
17:  int main(){
18:    par // v1, segment 1 starts
19:    b1.main();
20:    b2.main();
21:  } // v2, segment 2 starts
22:  printf("summation 1 is :%d \n", sum1);
23:  printf("summation 2 is :%d \n", sum2); }};
    
```

seg	<u>0</u>	<u>1</u>	<u>2</u>	<u>3</u>	<u>4</u>
<u>0</u>	F	F	F	F	F
<u>1</u>	F	F	F	T	T
<u>2</u>	F	F	F	T	T
<u>3</u>	F	T	T	T	T
<u>4</u>	F	T	T	T	T

➤ E.g.: Seg3 and Seg4 cannot run in parallel out of the order

(c) 2012 R. Doemer, et al. 28

OoO PDES Conflicts Checking

- Scheduling: Fast checking at runtime
 - Timestamp Comparison
 - Segment Conflict Table Lookup
- Timestamp Comparison (**th1** running, **th2** Candidate)
 - **th1.timestamp** = **th2.timestamp**, as regular PDES
 - **th1.timestamp** > **th2.timestamp**, issue **th2**
 - **th1.timestamp** > **th2.timestamp**, Check Conflicts
- Conflict Prediction
 - **th1** delivery an event and bring thread **th3** with earlier timestamps.
 - **th1** will be followed by a new segment with earlier timestamp than **th2**.
 - Build tables to lookup at runtime.

EECS Colloquium, October 17, 2012

(c) 2012 R. Doemer, et.al.

29

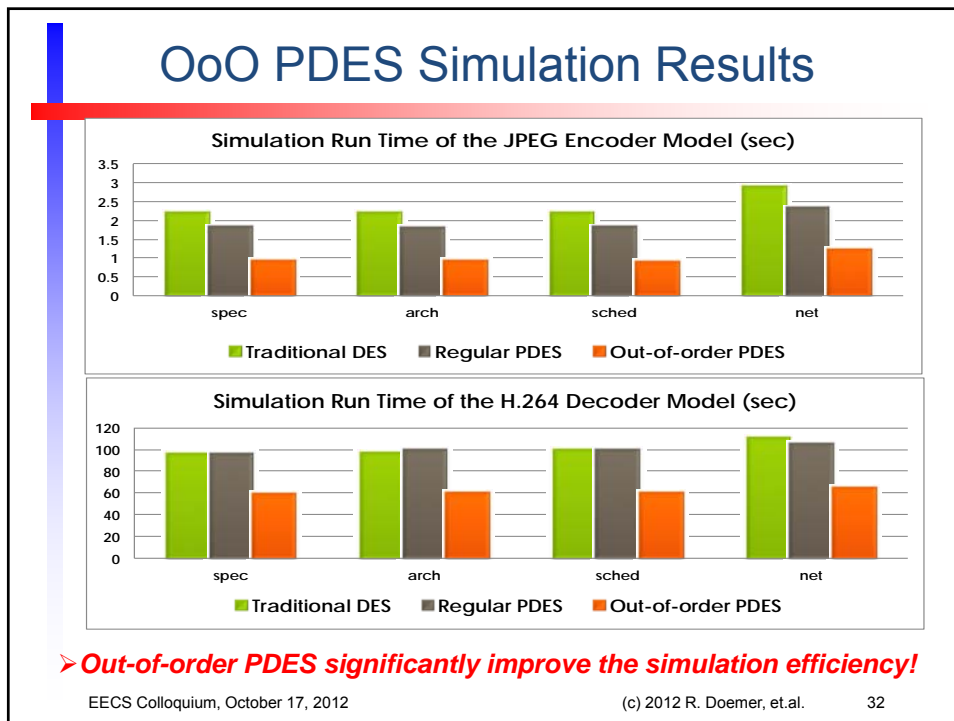
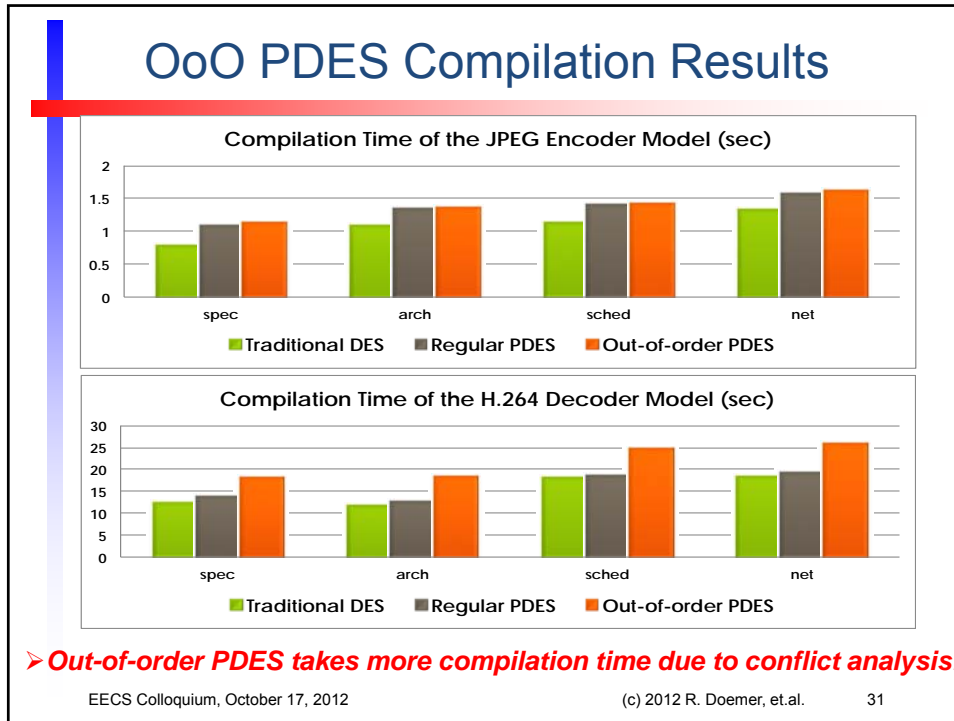
OoO PDES Experiments and Results

- Parallelized JPEG Image Encoder
 - 3 color components encoded in parallel
 - Sequential Huffman encoding
 - A color image with 3216x2136
- Parallelized H.264 Video Decoder
 - 4 parallel slice decoders
 - Sequential slice reader and synchronizer
 - Communication via double handshake channels
 - Video frames divided into 4 independent slices
 - 1079 frames (about 36 seconds of video) and 1280x720 pixels per frame
- **Results measured on a host PC with a 4-core CPU (Intel® Core™ Quad) at 3.0 GHz**

EECS Colloquium, October 17, 2012

(c) 2012 R. Doemer, et.al.

30



Concluding Remarks

- PDES can significantly speed up System Simulation!
- What is the Potential of Parallelism we can exploit?
 - SMP Host Platforms
 - Multi-core PCs are readily available, many-core platforms coming
 - Nearly unlimited potential!
 - Parallelism in Application sets an upper bound!
 - Designer needs to identify and explicitly state the parallelism
 - Need more research on effective parallelization!
- Out-of-Order PDES
 - Aggressive, breaks the cycle barrier, but preserves accuracy
 - Limited by the quality of static analysis in the compiler
 - Open question: How many *real* dependencies do exist?
 - The better the analysis, the less false dependencies!
 - Need more research on advanced static analysis!

EECS Colloquium, October 17, 2012

(c) 2012 R. Doemer, et.al.

33

References

- [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] R. Dömer, W. Chen, X. Han: "Parallel Discrete Event Simulation of Transaction Level Models", Proceedings of ASPDAC, Sydney, Australia, February 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.
- [CECS-TR'12] W. Chen, R. Dömer: "A Distributed Parallel Simulator for Transaction Level Models with Relaxed Timing", CECS TR 11-02, May 2011.
- [IEEE D&T'11] W. Chen, X. Han, R. Dömer: "Multicore Simulation of Transaction-Level Models Using the SoC Environment", IEEE Design & Test of Computers, vol. 28, no. 3, pp. 20-31, May-June 2011.
- [ASPDAC'11] R. Dömer, W. Chen, X. Han, A. Gerstlauer: "Multi-Core Parallel Simulation of System-Level Description Languages", Proceedings of ASPDAC, Yokohama, Japan, January 2011.
- [HLDVT'10] W. Chen, X. Han, R. Dömer: "ESL Design and Multi-Core Validation using the System-on-Chip Environment", Proceedings of HLDVT, Anaheim, California, June 2010.

EECS Colloquium, October 17, 2012

(c) 2012 R. Doemer, et.al.

34