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

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

UNSW, 2 March 2016

(c) 2016 R. Doemer, CECS 3

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

Out-of-Order Parallel SystemC, UNSW, 2 March 2016 (c) 2016 R. Doemer, CECS 4

Project Context: Related Work

Modeling Techniques
- Transaction-level modeling (TLM).
- TLM temporal decoupling.
- Savoiu et al. [MEMOCODE’05]
- Razaghi et al.[ASPDAC’12]

Discrete Event Simulation is slow

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

Distributed Simulation
- Chandy et al. [TSE’79]
- Huang et al. [SIES’08]
- Chen et al. [CECS’11]

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]
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* \([\text{DATE}'12, \text{TCAD}'14]\) 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*!

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
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*)
  - 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 PDES Technology
Out-of-Order Parallel Simulation of SystemC Models on Many-Core Platforms

UNSW, 2 March 2016

(c) 2016 R. Doemer, CECS

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 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
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
Dedicated SystemC Compiler

• RISC Software Stack
  - Recoding Infrastructure for SystemC
    - SystemC Internal Representation
  - Class hierarchy to represent SystemC objects

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

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

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

```c
void straight() {
    x = 42;
    int xx = 43;
    int yy;
    int o = y;
    wait(10, SC_NS);
    wait();
    int kk;
    wait();
    int oo;
}
```

<table>
<thead>
<tr>
<th>Segment ID: 0</th>
<th>input_straight.cpp:24 (this) -&gt; x = 42</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>input_straight.cpp:25 int xx = 43;</td>
</tr>
<tr>
<td></td>
<td>input_straight.cpp:26 int yy;</td>
</tr>
<tr>
<td></td>
<td>input_straight.cpp:27 yy</td>
</tr>
<tr>
<td></td>
<td>input_straight.cpp:28 int o = (this) -&gt; y;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Segment ID: 1</th>
<th>input_straight.cpp:30</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Segment ID: 2</th>
<th>input_straight.cpp:32</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Segment ID: 3</th>
<th>input_straight.cpp:37</th>
</tr>
</thead>
</table>

---

### Dedicated SystemC Compiler

**Segment Graph Construction:**
- Support for conditional statements

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

<table>
<thead>
<tr>
<th>Segment ID: 0</th>
<th>input_if_else.cpp:27</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Segment ID: 1</th>
<th>input_if_else.cpp:28 int aaa; compilerGenerated:0 (this) -&gt; test</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Segment ID: 2</th>
<th>input_if_else.cpp:30 int bbb;</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Segment ID: 3</th>
<th>input_if_else.cpp:34 int ddd;</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Segment ID: 4</th>
<th>input_if_else.cpp:35</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Segment ID: 5</th>
<th>input_if_else.cpp:36 int eee;</th>
</tr>
</thead>
</table>

---

(c) 2016 R. Doemer, CECS
Dedicated SystemC Compiler

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

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

```c
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
    ➢ \( f(x), \text{return} \)

```c
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;
}
```

Segment ID: 1 (input_function_calls.cpp:152)
Segment ID: 2 (input_function_calls.cpp:163)
Segment ID: 3 (input_function_calls.cpp:169)
Segment ID: 4 (input_function_calls.cpp:176)
Segment ID: 0

Segment ID: 0
Segment ID: 1 (input_function_calls.cpp:151)
Segment ID: 2 (input_function_calls.cpp:159)
Segment ID: 3 (input_function_calls.cpp:161)
Segment ID: 4 (input_function_calls.cpp:173)
Segment ID: 5 (input_function_calls.cpp:177)
Dedicated SystemC Compiler

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

Instrumentation!

Segment Graph Construction

Parallel Access Conflict Analysis

SystemC Model

- systemc.h
- Model.cpp

Parallel C++ Model

Model.cpp

Compilation, Simulation

RISC

SystemC IR

ROSE IR

C/C++ Foundation

**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
Dedicated SystemC Compiler

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

```cpp
class Conflict: public sc_module {
SC_CTOR(Conflict)
{ SC_THREAD(thread1);
SC_THREAD(thread2);
}
int x, y, z;
void thread1() { int a;
wait();
a = 2;
wait();
a = x + y;
wait();
z++;
};
void thread2() { int b = 2;
x = y;
wait();
x = y * z;
wait();
z++;
wait();
z++;
};
```

Segment Graph

Segment ID: 0
conflict.cpp:24 int a;
conflict.cpp:25 a = 2
Segment ID: 3
conflict.cpp:34 int b = 2;
conflict.cpp:35 x = y
Segment ID: 1 (conflict.cpp:26)
conflict.cpp:27 a = x + y
Segment ID: 2 (conflict.cpp:28)
conflict.cpp:29 z++
Segment ID: 4 (conflict.cpp:36)
conflict.cpp:37 x = y * z
Segment ID: 5 (conflict.cpp:38)
conflict.cpp:39 z++
Segment ID: 6 (conflict.cpp:40)
conflict.cpp:41 z++
Dedicated SystemC Compiler

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

  ```
  Segment ID: 0
  (W) x
  (R) y
  Segment ID: 1 (par/0.cpp:26)
  (W) x
  (R) y
  (R) z
  Segment ID: 2 (par/0.cpp:26)
  (W) x
  Segment ID: 3
  (W) x
  (R) y
  Segment ID: 4 (par/0.cpp:38)
  (W) x
  (R) y
  (R) z
  Segment ID: 5 (par/0.cpp:38)
  (W) x
  Segment ID: 6 (par/0.cpp:46)
  (W) x
  ```

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

  Segment Data Conflict Table

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)
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

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)
Example Model 1: DVD Player

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

```c
1: SC_MODULE(VideoCodec)
2: { sc_port<i_receiver> p1;  
3:   sc_port<i_sender> p2;  
4:   ...  
5:   while(1){ 
6:     p1->receive(&inFrm); 
7:     outFrm = decode(inFrm); 
8:     wait(33330, SC_US); 
9:     p2->send(outFrm);  
10:   }  
11: ];
```

```c
1: SC_MODULE(AudioCodec)
2: { sc_port<i_receiver> p1;  
3:   sc_port<i_sender> p2;  
4:   ...  
5:   while(1){ 
6:     p1->receive(&inFrm); 
7:     outFrm = decode(inFrm); 
8:     wait(26120, SC_US); 
9:     p2->send(outFrm);  
10: }  
11: ];
```

Example Model 1: DVD Player

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

1. Real time schedule: fully parallel

<table>
<thead>
<tr>
<th>Frame</th>
<th>Time [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>26.12</td>
</tr>
<tr>
<td>3</td>
<td>52.25</td>
</tr>
<tr>
<td>4</td>
<td>78.38</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
</tr>
</tbody>
</table>

1. Reference simulator schedule (DES)

<table>
<thead>
<tr>
<th>Frame</th>
<th>Time [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>26.12</td>
</tr>
<tr>
<td>3</td>
<td>52.25</td>
</tr>
<tr>
<td>4</td>
<td>78.38</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
</tr>
</tbody>
</table>
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. Out-of-order parallel schedule (OoO PDES)
  3. Synchronous parallel schedule (PDES)

Out-of-Order Parallel Simulation of SystemC Models on Many-Core Platforms
UNSW, 2 March 2016
(c) 2016 R. Doemer, CECS
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

<table>
<thead>
<tr>
<th></th>
<th>Seq</th>
<th>Par</th>
<th>OoO</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 sec stream</td>
<td>Run Time</td>
<td>6.98 s</td>
<td>4.67 s</td>
</tr>
<tr>
<td></td>
<td>CPU Load</td>
<td>97%</td>
<td>145%</td>
</tr>
<tr>
<td></td>
<td>Speedup</td>
<td>1 x</td>
<td>1.49 x</td>
</tr>
<tr>
<td>100 sec stream</td>
<td>Run Time</td>
<td>68.21 s</td>
<td>45.91 s</td>
</tr>
<tr>
<td></td>
<td>CPU Load</td>
<td>100%</td>
<td>149%</td>
</tr>
<tr>
<td></td>
<td>Speedup</td>
<td>1 x</td>
<td>1.49 x</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parallel Slices</th>
<th>DES Run Time</th>
<th>CPU Load</th>
<th>PDES Run Time</th>
<th>CPU Load</th>
<th>Speedup</th>
<th>OOO PDES Run Time</th>
<th>CPU Load</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>162.13 s</td>
<td>99%</td>
<td>162.06 s</td>
<td>100%</td>
<td>1.00 x</td>
<td>161.90 s</td>
<td>100%</td>
<td>1.00 x</td>
</tr>
<tr>
<td>2</td>
<td>162.19 s</td>
<td>99%</td>
<td>96.50 s</td>
<td>168%</td>
<td>1.68 x</td>
<td>96.48 s</td>
<td>168%</td>
<td>1.68 x</td>
</tr>
<tr>
<td>4</td>
<td>162.56 s</td>
<td>99%</td>
<td>54.00 s</td>
<td>305%</td>
<td>3.01 x</td>
<td>53.85 s</td>
<td>304%</td>
<td>3.02 x</td>
</tr>
<tr>
<td>8</td>
<td>163.10 s</td>
<td>99%</td>
<td>29.89 s</td>
<td>592%</td>
<td>5.46 x</td>
<td>30.05 s</td>
<td>589%</td>
<td>5.43 x</td>
</tr>
<tr>
<td>16</td>
<td>164.01 s</td>
<td>99%</td>
<td>19.03 s</td>
<td>1050%</td>
<td>8.62 x</td>
<td>20.08 s</td>
<td>997%</td>
<td>8.17 x</td>
</tr>
<tr>
<td>32</td>
<td>165.89 s</td>
<td>99%</td>
<td>11.78 s</td>
<td>2082%</td>
<td>14.08 x</td>
<td>11.99 s</td>
<td>2023%</td>
<td>13.84 x</td>
</tr>
<tr>
<td>64</td>
<td>170.32 s</td>
<td>99%</td>
<td>9.79 s</td>
<td>2607%</td>
<td>17.40 x</td>
<td>9.85 s</td>
<td>2608%</td>
<td>17.29 x</td>
</tr>
<tr>
<td>128</td>
<td>174.55 s</td>
<td>99%</td>
<td>9.34 s</td>
<td>2793%</td>
<td>18.69 x</td>
<td>9.39 s</td>
<td>2787%</td>
<td>18.59 x</td>
</tr>
<tr>
<td>256</td>
<td>185.47 s</td>
<td>100%</td>
<td>8.91 s</td>
<td>2958%</td>
<td>20.82 x</td>
<td>8.90 s</td>
<td>2964%</td>
<td>20.84 x</td>
</tr>
</tbody>
</table>
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 cache
    • 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