Advances in Parallel Discrete Event Simulation for Embedded Computer Systems

Rainer Dömer, Professor EECS
doemer@uci.edu

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

Embedded System Design

- Model Based System Design
  - 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
Embedded System 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]: \texttt{SC\_THREAD}, \texttt{SC\_METHOD}
      - SpecC [Gajski et. al, 2000]: \texttt{par \{\}, pipe \{\}}
  - Parallel processing available in standard PCs
    - Multi-core host PCs readily available
    - Many-core technology is arriving

Related Work: Faster Simulation

Improved 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]

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. [TCAD'14]
- Yun et al. [TCAD'12]
- Schmidt et al. [DAC'17]
- and many others

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

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, 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 Sequential Simulation**
  - 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 1666 Standard Simulator**
  - SystemC reference simulator
    - Uses cooperative multi-threading
    - A single thread is active at any time!
    - Cannot exploit parallelism
    - Cannot utilize multiple cores
    - Sequential simulation is slow

Parallel Discrete Event Simulation (PDES)

- **Parallel DES [Fujimoto1990]**
  - Threads execute in parallel \( \text{iff} \)
    - in the same delta cycle, \text{and}
    - in the same time cycle
  - **Significant speed up!**
  - **Synchronous PDES:**
    - Cycle boundaries are \text{absolute barriers}!
- **Aggressive Parallel DES**
  - **Conservative Approaches**
    - Careful static analysis prevents conflicts
  - **Optimistic Approaches**
    - Conflicts are detected and addressed \( \text{(roll back)} \)
Parallel Discrete Event Simulation (PDES)

- **Out-of-Order PDES**
  - Threads execute in parallel *iff*
  - in the same delta cycle, *and*
  - in the same time cycle,
  - *OR if there are no conflicts!*
  - Breaks synchronization barrier!
  - Threads run as soon as possible, even ahead of time
  - Significantly higher speedup!
  - Results at [DATE’12], [IEEE TCAD’14]
  - Advanced compiler fully preserves…
    - DES execution semantics
    - Accuracy in results and timing

Recoding Infrastructure for SystemC (RISC)

- **Advanced Parallel SystemC Simulation**
  - Aggressive PDES on many-core host platforms
  - Maximum compliance with IEEE SystemC semantics
- **Introduction of a Dedicated SystemC Compiler**
  - Advanced conflict analysis for safe parallel execution
  - Automatic model instrumentation and code generation
- **Parallel SystemC Simulator**
  - Out-of-order parallel scheduler, multi-thread safe primitives
  - Multi- and many-core host platforms (e.g. Intel® Xeon Phi™)
- **Open Source**
  - Freely available for evaluation and collaboration
  - Thanks to Intel Corporation!
Recoding Infrastructure for SystemC (RISC)

- Out-of-Order PDES Key Ideas
  1. Dedicated SystemC compiler with advanced model analysis
     - Static conflict analysis based on Segment Graphs
  2. Parallel simulator with out-of-order scheduling 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 article: “OoO PDES for TLM” [IEEE TCAD’14]
    - Comprehensive summary with HybridThreads extension

RISC: Dedicated SystemC Compiler

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

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

Source: Lawrence Livermore National Laboratory (LLNL)
RISC: Dedicated SystemC Compiler

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

RISC

SystemC IR

ROSE IR

C/C++ Foundation

SystemC Compiler

- Segment Graph
- Parallel access conflict analysis

SystemC Model

Model.cpp

Segment Graph

Step 1: Build a Segment Graph

PDES for Embedded Systems, Chapman University, 4/25/23
(c) 2023 R. Doemer, UC Irvine
RISC: Dedicated SystemC Compiler

• Segment Graph
  – Segment Graph is a directed graph
    • Nodes: Segments
      ➢ Code statements executed between two scheduling steps
        – Expression statements
        – Control flow statements (if, while, …)
        – Function calls
    • Edges: Segment boundaries
      ➢ Primitives that trigger scheduler entry
        – wait(event)
        – wait(time)
  ➢ Segment Graph is built automatically by the compiler [TCAD’14]
    • From the model source code
    • Via Abstract Syntax Tree and Control Flow Graph

RISC: Dedicated SystemC Compiler

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

SystemC Model
  – systemc.h
  – Model.cpp

SystemC Compiler
  – Segment Graph Construction
  – Parallel Access Conflict Analysis

RISC
  – Segment Graph
  – SystemC IR
  – ROSE IR
  – C/C++ Foundation

Compilation, Simulation

Parallel C++ Model
  – Model_per.cpp

Instrumentation!

Seg 1
Seg 2
Seg 3
Seg 4
Seg 5
Seg 6
Segment Graph

Seg 1
Seg 2
Seg 3
Seg 4
Seg 5
Seg 6
Segment Graph

R: a, b
W: x, y
RW: z

R: a, b
W: x, y
RW: z

Conflicts:

Seg 1: True
Seg 2: True
Seg 3: True
RISC: Compiler and Simulator

- Compiler and Simulator work hand in hand
  - Compiler performs conservative static analysis
  - Analysis results are passed to the simulator
  - Simulator can make safe scheduling decisions quickly

- Automatic Model Instrumentation
  - Static analysis results are inserted into the source code

RISC Simulator

- C++ Compiler
- Out-of-Order Parallel Simulation

SystemC Compiler

- Parallel C++ Model
- Model _par.cpp

Model Instrumentation:
- Segment and Instance IDs
- Segment Conflict Tables
- Time Advance Tables

RISC: Parallel SystemC Simulator

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

- Issue threads...
  - truly in parallel and out-of-order
  - whenever they are ready
  - and have no conflicts!
  - Fast conflict table lookup
  - Optimized thread-to-core mapping
RISC: Experiments and Results

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

1. Real time schedule: fully parallel

2. Reference simulator schedule (DES)

(c) 2023 R. Doemer, CECS, UC Irvine
RISC: Experiments and Results

- DVD Player Example
  - 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)
RISC: Experiments and Results

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

<table>
<thead>
<tr>
<th></th>
<th>DES</th>
<th>PDES</th>
<th>OxO PDES</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 sec stream</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Run Time</td>
<td>6.98 s</td>
<td>4.67 s</td>
<td>2.94 s</td>
</tr>
<tr>
<td>CPU Load</td>
<td>97%</td>
<td>145%</td>
<td>238%</td>
</tr>
<tr>
<td>Speedup</td>
<td>1 x</td>
<td>1.49 x</td>
<td>2.37 x</td>
</tr>
<tr>
<td>100 sec stream</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Run Time</td>
<td>68.21 s</td>
<td>45.91 s</td>
<td>28.13 s</td>
</tr>
<tr>
<td>CPU Load</td>
<td>100%</td>
<td>149%</td>
<td>251%</td>
</tr>
<tr>
<td>Speedup</td>
<td>1 x</td>
<td>1.49 x</td>
<td>2.42 x</td>
</tr>
</tbody>
</table>

RISC: Experiments and Results

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

- Mandelbrot Renderer (Graphics Pipeline Application)
  - Simulator run times on 16-core Intel® Xeon® multi-core host
  - 2 CPUs at 2.7 GHz, 8 cores each, 2-way hyper-threaded
  - RISC V0.2.1, Posix-threads

<table>
<thead>
<tr>
<th>Parallel Slices</th>
<th>DES Run Time</th>
<th>DES CPU Load</th>
<th>PDES Run Time</th>
<th>PDES CPU Load</th>
<th>Speedup</th>
<th>OOO PDES Run Time</th>
<th>OOO PDES 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>2956%</td>
<td>20.82 x</td>
<td>8.90 s</td>
<td>2964%</td>
<td>20.84 x</td>
</tr>
</tbody>
</table>
RISC: Experiments and Results

- Many-Core Target Platform: Intel® Xeon Phi™
  - Many Integrated Core (MIC) architecture
    - 1 Coprocessor 5110P CPU at 1.052 GHz
    - 60 physical cores with 4-way hyper-threading
    - Appears as regular Linux host with 240 cores
    - Up to 8 lanes available for vector processing
  - RISC extended for exploiting 2 types of parallelism
    - Out-of-Order PDES: thread-level parallelism
    - Intel® compiler SIMD: data-level parallelism
  - DAC '17 paper: “Exploiting Thread and Data Level Parallelism for Ultimate Parallel SystemC Simulation”

- Experimental Results:
  - Mandelbrot renderer (graphics pipeline application)
  - Increasing degree of parallelism (PAR = number of threads) reaches a combined multi-threading (MT) and data-level (SIMD) speedup of up to 212x!
RISC: Open Source Software

- RISC Compiler and Simulator are freely available
  - http://www.cecs.uci.edu/~doemer/risc.html#RISC062
    - Installation notes and script: INSTALL, Makefile
    - Open source tar ball: risc_v0.6.3.tar.gz
    - Docker script and container: Dockerfile
    - Doxygen documentation: RISC API, OOPSC API
    - Tool manual pages: risc, simd, visual,
    - BSD license terms: LICENSE
  - Companion Technical Report

  Docker container:
  - https://hub.docker.com/r/ucirvinelecs/risc063/

bash# docker pull ucirvinelecs/risc063
bash# docker run -it ucirvinelecs/risc063
[dockeruser]# cd demodir
[dockeruser]# make test

Conclusion

- Recoding Infrastructure for SystemC (RISC)
  - Out-of-Order Parallel SystemC Simulation
    - Aggressive PDES on many-core host platforms
    - Maximum compliance with IEEE SystemC semantics
  - Introduction of a Dedicated SystemC Compiler
    - Advanced conflict analysis for safe parallel execution
    - Automatic model instrumentation and code generation
  - Parallel SystemC Simulator
    - Out-of-order parallel scheduler, multi-thread safe primitives
    - Multi- and many-core host platforms (e.g. Intel® Xeon Phi™)
  - Open Source
    - Freely available for use and collaboration (BSD license)
    - Thanks to Intel Corporation!
Acknowledgments

• For solid work, fruitful discussions, and honest feedback, I would like to thank:
  – My team at UCI
    • Emad Arasteh, Aditya Harit, Vivek Govindasamy
    • Zhongqi Cheng, Daniel Mendoza
    • Tim Schmidt, Guantao Liu
    • Farah Arabi, Spencer Kam
  – Our collaborators at Intel
    • Ajit Dingankar
    • Desmond Kirkpatrick
    • Abhijit Davare
    • Philipp Hartmann
  – And many others…
• This work has been supported in part by substantial funding from Intel Corporation. Thank you!

References (1)

References (2)


References (3)

RISC Open Source Releases

  *RISC Compiler and Simulator, Release V0.6.3.* August 2021. 
  - [http://cecs.uci.edu/~doemer/risc.html#RISC063](http://cecs.uci.edu/~doemer/risc.html#RISC063)
  *RISC Compiler and Simulator, Release V0.6.2.* September 2020.
- [RISCv060] Z. Cheng, R. Dömer, S. Kam, and D. Mendoza. 
  *RISC Compiler and Simulator, Release V0.6.0.* September 2019.
  *RISC Compiler and Simulator, Release V0.5.0.* September 2018.
  *RISC Compiler and Simulator, Release V0.4.0.* July 2017.
  *RISC Compiler and Simulator, Beta Release V0.3.0.* September 2016.
  *RISC Compiler and Simulator, Alpha Release V0.2.0.* September 2015.
  *RISC API, Alpha Release V0.1.0.* June 2014.