OpenTimer: A high-performance timing analysis tool

LibreCores Student Design Contest

Tsung-Wei Huang, Chun-Xun Lin, and Martin D. F. Wong
Department of Electrical and Computer Engineering (ECE)
University of Illinois at Urbana-Champaign (UIUC), IL, USA
Motivation of OpenTimer Project – TAU Contests

- An open-source STA engine with incremental timing and CPPR
  - Important for timing-driven applications
  - Fast full timing and incremental timing analysis
  - Capability of path-based CPPR analysis
  - Parallel programming and multi-threading

*CPPR stands for Common Path Pessimism Removal*
OpenTimer Architecture

- An open-source high-performance timing analysis tool
  - TAU14 (1st place), TAU15 (2nd place), TAU16 (1st place)
  - Selected as the golden timer in ICCAD15, TAU16, and TAU17 contests

- Feature highlights
  - C++11
  - Industry format
  - STA engine
  - Block-based
  - Path-based
  - Incremental
  - Lazy evaluation
  - CPPR
  - Multi-threaded

http://web.engr.illinois.edu/~thuang19/software/timer/OpenTimer.html
# Experimental Results – Overall Performance Comparison

## Table I

<table>
<thead>
<tr>
<th>Circuit</th>
<th>#Gates</th>
<th>#Nets</th>
<th>#OPs</th>
<th>iitRACE</th>
<th>iTimerC 2.0</th>
<th>OpenTimer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>accuracy</td>
<td>runtime</td>
<td>memory</td>
</tr>
<tr>
<td>b19</td>
<td>255.3K</td>
<td>255.3K</td>
<td>5641.5K</td>
<td>63.03 %</td>
<td>629 s</td>
<td>3.0 GB</td>
</tr>
<tr>
<td>cordic</td>
<td>45.4K</td>
<td>45.4K</td>
<td>1607.6K</td>
<td>61.83 %</td>
<td>100 s</td>
<td>0.9 GB</td>
</tr>
<tr>
<td>des_perf</td>
<td>138.9K</td>
<td>139.1K</td>
<td>4326.7K</td>
<td>67.43 %</td>
<td>299 s</td>
<td>4.2 GB</td>
</tr>
<tr>
<td>edit_dist</td>
<td>147.6K</td>
<td>150.2K</td>
<td>3368.3K</td>
<td>64.83 %</td>
<td>857 s</td>
<td>2.0 GB</td>
</tr>
<tr>
<td>fft</td>
<td>38.2K</td>
<td>39.2K</td>
<td>1751.7K</td>
<td>89.66 %</td>
<td>70 s</td>
<td>0.5 GB</td>
</tr>
<tr>
<td>leon2</td>
<td>1616.4K</td>
<td>1517.0K</td>
<td>8438.5K</td>
<td>72.34 %</td>
<td>16832 s</td>
<td>9.9 GB</td>
</tr>
<tr>
<td>leon3mp</td>
<td>1247.7K</td>
<td>1248.0K</td>
<td>8405.9K</td>
<td>62.99 %</td>
<td>4960 s</td>
<td>8.2 GB</td>
</tr>
<tr>
<td>mgc_edit_dist</td>
<td>161.7K</td>
<td>164.2K</td>
<td>3403.4K</td>
<td>64.29 %</td>
<td>1578 s</td>
<td>1.9 GB</td>
</tr>
<tr>
<td>mgc_matrix_mult</td>
<td>171.3K</td>
<td>174.5K</td>
<td>3717.5K</td>
<td>67.93 %</td>
<td>1363 s</td>
<td>2.0 GB</td>
</tr>
<tr>
<td>netcard</td>
<td>1496.0K</td>
<td>1497.8K</td>
<td>11594.6K</td>
<td>87.63 %</td>
<td>6662 s</td>
<td>9.4 GB</td>
</tr>
<tr>
<td>cordic_core</td>
<td>3.6K</td>
<td>3.6K</td>
<td>226.0K</td>
<td>59.42 %</td>
<td>21 s</td>
<td>0.3 GB</td>
</tr>
<tr>
<td>crc32d16N</td>
<td>478</td>
<td>495</td>
<td>28.9K</td>
<td>57.15 %</td>
<td>3 s</td>
<td>0.1 GB</td>
</tr>
<tr>
<td>softusb-navre</td>
<td>6.9K</td>
<td>7.0K</td>
<td>427.8K</td>
<td>40.17 %</td>
<td>21 s</td>
<td>0.1 GB</td>
</tr>
<tr>
<td>tip_master</td>
<td>37.7K</td>
<td>38.5K</td>
<td>1300.4K</td>
<td>82.95 %</td>
<td>64 s</td>
<td>0.6 GB</td>
</tr>
<tr>
<td>vga_lcd_1</td>
<td>139.5K</td>
<td>139.6K</td>
<td>2961.5K</td>
<td>99.65 %</td>
<td>260 s</td>
<td>1.6 GB</td>
</tr>
<tr>
<td>vga_lcd_2</td>
<td>259.1K</td>
<td>259.1K</td>
<td>12674.7K</td>
<td>98.57 %</td>
<td>1132 s</td>
<td>13.3 GB</td>
</tr>
</tbody>
</table>

*#Gates: number of gates. #Nets: number of nets. #OPs: number of operations. accuracy: average of path accuracy and value accuracy (%). -: program crash.

*iTimerC 2.0: IEEE/ACM ICCAD15 (binary from authors)
*iitRACE: IEEE/ACM ICCAD15 (binary from authors)

Golden reference by IBM Einstimer
Experimental Results – Scalability of Incremental Timing

Scalability (vga_lcd_2)

- OpenTimer
- iTimerC 2.0

Total runtime (s) vs. # incremental processing stages

X115 speedup by OpenTimer (at 1459th stage)

X2.7 speedup by OpenTimer (at 1th stage, i.e. full timing)

Parallel IO by OpenTimer

Parallel timing by OpenTimer
Conclusion

- Developed a high-performance timing analysis tool
  - Free software and open-source under GPL v3.0
  - Industry format (.v, .spef, .lib, .lef, .def, etc.)
  - Fast, accurate, robust, and CPPR by default

- Recognitions and contributions to community
  - 1st prize in TAU14 contest (full timing with CPPR)
  - 2nd prize in TAU15 contest (incremental timing with CPPR)
  - 1st prize in TAU16 contest (timing macro-modeling)
  - Golden timer in ICCAD15 CAD contest
  - Golden timer in TAU16 contest
  - Golden timer in TAU17 contest

- Acknowledgment
  - Jin, Billy, M.-C., team iTimerC, team iitRACE, and the UIUC CAD group!
Backup slides
Initialize the Timer – Parallel IO

- A set of design files that follow the industry standard format
  - Verilog netlist, two libraries (early and late), parasitic spef, etc.
  - Time-consuming IO (e.g., file IO, complex parsing)

Input files (industry standard)

- Verilog (.v)
- Liberty (.lib)
- SPEF (.spef)

- Assertions (.timing)
- Operations (.ops)

Based on TAU 2013, ISPD 2013, ICCAD 2014, Cadence Benchmarks

Task graph

Library (Early) → Library (Late)

Verilog

Parasitic

LEF

DEF

Assertion

Parallel dependency generation using portable OpenMP

#pragma parallel ...
#pragma task ...

x2 speedup by parallel read/parse!
Timing Graph Reduction

- Reduce the search space
  - Identify tree-structured subgraphs in the original timing graph
  - Merge every leaf-root path (transition-definite)

Every leaf-root path can be uniquely defined (given a transition at an endpoint)

- Approximately 30% reduction on the graph size
- Apply to combinational circuits only
Key Components of Incremental Timing

- Full timing is just a special case of incremental timing
- Design modifiers
  - Pin-level operations, net-level operations, and gate-level operations
- Timing queries
  - Slack
  - Arrival time
  - Required time
  - TNS and WNS
  - Critical path report
  - CPPR
- Source of propagation
- Lazy evaluation
- Explore parallel incremental timing
Common path pessimism removal (CPPR)

- Constant-space and -time representation for CPPR

(a) Build the suffix graph: shortest distance to target
(b) Spur along the 1st critical path (post-CPPR = -12)
(c) Spur along the 2nd critical path (post-CPPR = -11)
(d) Spur along the 3rd critical path (post-CPPR = -10)
(e) Spur along the 4th critical path (post-CPPR = -8)
(f) Spur along the 6th critical path (post-CPPR = -4)
Pipeline-based Parallel Timing Propagation

- Timing propagation has several linearly dependent tasks
  - RC update → Slew & Delay → Arrival time → Jump point → CPPR
  - Pipeline scheduling with multiple threads

We use the following paper for dealing with CPPR
*UI-Timer: An ultra-fast clock network pessimism removal algorithm, T.-W. Huang, P.-C. Wu, and Martin D. F. Wong, ICCAD14*