Statistical Path Tracing in Timing Graphs

Vasant Rao (IBM Poughkeepsie, NY)
Debjit Sinha (IBM Poughkeepsie, NY)
Nitin Srimal (IBM Bangalore, India)
Prabhat Maurya (IBM Bangalore, India)

March 10, 2016
Background

• Path tracing is a key step in **static timing analysis and optimization**
  – Given a (set of) timing end-point(s) in a graph, trace **backward** and find a set of most critical paths leading to each end-point
  – Applications
    • Reporting (top N critical paths)
    • Common path pessimism reduction (CPPR)
    • Optimization of a set of most critical paths

• In a **deterministic** timing environment, backward path tracing is performed using notion of critical arrival time (AT)
  – In this case, path tracing using AT is equivalent to tracing using slack

• In a **statistical** timing environment, backward path tracing is **not a simple extension** of path tracing during deterministic timing

• Possible approaches to path tracing
  – **DPT**: (Deterministic Path Tracing) Based on deterministic AT or slack – **Incorrect** (example follows)
  – Based on statistical AT (projected, tightness-probability based, etc.) – **Incorrect** (example follows)

• Statistical required arrival time (RAT) has variability, plays a role in which path(s) are most critical (and at what corner)

• **SPT**: Statistical (slack based) Path Tracing is the correct way
  – However, how to rank order paths? Criticality – Definition?
Linear Canonical Form (LCF)

\[ a_0 + a_1 \Delta X_1 + a_2 \Delta X_2 + \cdots + a_N \Delta X_N + a_R \Delta R \]

- Mean (nominal value)
- Sensitivities
- Deviation of global sources of variation from their nominal values
- Random uncertainty (deviation from nominal value)

What does this canonical form have to do with statistical timing?
- determining the \( a_1, a_2, \ldots, a_N \) and \( a_R \) coefficients is sensitivity analysis
- but if you think of \( \Delta X_1, \Delta X_2, \ldots, \Delta X_N, \Delta R \) as random Gaussian variables, then this result is a statistical result (or linear regression) which can be plotted as a distribution
LCF Math Operations

• Stat Add/Subtract
  – Mean and regular sensitivities are simply added/subtracted
  – Random sensitivity is always RSSed (root sum squared)

• Stat Min/Max
  – Clark’s linear Gaussian approximation using tightness probabilities is used
  – Often introduces additional random term

• Projection: Get a number out of a Distribution (LCF)
  – Given corner $c = \{c_1, c_2, ..., c_N, c_R\}$
  – Define $Proj_c(A) = a_0 + \sum_{i=1}^{N} a_i c_i + a_R c_R$
  – Often $c_i = \{-3, +3\}$. Also, sensitivities are usually RSSed to reduce pessimism.

• For simplicity, we will assume only simple LCFs in this paper.
  – All Timing quantities such as Delay, Arrival Time (AT), Slew, Required Arrival Time (RAT), and Slack will be assumed to be simple LCFs.

$Slack = \begin{cases} 
RAT - AT \text{ if Late Mode} \\
AT - RAT \text{ if Early Mode} 
\end{cases}$
Late Mode Test Example

- **Late Mode Stat Arrival Times:**
  
  \[ AT_1 = 99 + 3.9V + 2.9T \]
  
  \[ AT_2 = 110 + 0.3V + 0.2T \]
  
  \[ AT_1 + d_1 = 100 + 4V + 3T \]
  
  \[ AT_2 + d_2 = 111 + 0.4V + 0.3T \]

- **Stat MAX at 3:** \( AT_3 = \text{MAX}(AT_1 + d_1, AT_2 + d_2) \)
  
  \[ AT_3 = 111.011 + 0.167R + 0.426V + 0.32T \]

  - **Tightness Probabilities:** \( tp_1 = 0.007254, \)
    \( tp_2 = 0.992746 \)
    
  - Closer to \( AT_2 + d_2 \). So, based on AT (Det. or Stat.), Node 2 is more “critical” than 1 feeding 3.
    
    - Random appears in MAX even though incoming ATs had no random

- **RAT on D:** \( RAT_D = 112.523 + 0.6V + 0.5T \)

- **Back-propagate RAT from D to 3** (fanout at 4 does not contaminate RATs at either 4 or 3):
  
  \[ RAT_D - d_4 - d_3 = RAT_3 = 110.523 + 0.4V + 0.3T \]

- **Slack = RAT – AT** (random is RSSed; hence positive):
  
  \[ Slk_D = Slk_3 = -0.488 + 0.167R - 0.026V - 0.02T \]

  \[ Proj(Slk_D) = -0.488 - 3 \times \sqrt{(0.167)^2 + (-0.026)^2 + (-0.02)^2} = -1 \]

  Using RSS in Projection here
DPT Method

• Work with Deterministic Numbers
  – Assume deterministic corner \((V, T) = (0,0)\)
  – So, deterministic = mean

• Start at Data End of the Test (D) with projected Slack: \(\text{Proj}(\text{Slk}_D) = -1\)

• Suppose CPPR cutoff = 0. So, start peeling paths for this test.

• Walk back to first “merge” point 3.

• Late Mode Incoming Deterministic Propagated ATs:
  \[\text{AT}_1 + d_1 = 100 \quad \text{Diff} = 11 = 111-100\]
  \[\text{AT}_2 + d_2 = 111 \quad \text{2 Wins}\]

• 2 wins because it has higher propagated AT.
  – Copy slack (-1) from 3 and back-propagate critical path to 2.

• Compute “slack” at 1 = sink_slack + AT_diff = -1 + 11 = +10
  – Would not schedule 1 for later peeling since it exceeds CPPR cutoff = 0.

• Problems with DPT:
  1. Chooses 2 to be “critical” and assumes \(\text{Proj}(\text{Slk}_2) = -1\). False slack continuity.
     ➢ Will see this to be incorrect.
  2. Does not schedule 1 for future peeling because it “thinks” \(\text{Proj}(\text{Slk}_1) = +10\).
     ➢ Will see this to be incorrect also.
Proposed SPT Method

- Start at Data End of the Test (D) with projected Slack canonical
  \[ \text{Proj}(\text{Slk}_D) = -1 \]
- Suppose CPPR cutoff = 0.
  - So, start peeling paths for this test.
- Walk back to first “merge” point 3.
- Compute Edge-Slacks: \( \text{RAT}_3 = 110.523 + 0.4V + 0.3T \)

\[
\begin{align*}
\text{Slk}_1 &= \text{RAT}_3 - (\text{AT}_1 + d_1) = 10.523 - 3.6V - 2.7T; \quad \text{Proj}(\text{Slk}_1) = -2.977 \\
\text{Slk}_2 &= \text{RAT}_3 - (\text{AT}_2 + d_2) = -0.477 + 0V + 0T; \quad \text{Proj}(\text{Slk}_2) = -0.477 \\
\end{align*}
\]

- 1 wins because it has lower projected edge-slapck = -2.977.
  - back-propagate critical path to 1.
- Schedule 2 for later peeling with projected edge-slapck = -0.477 < 0.
- Note that DPT got both slacks at 1 and 2 wrong.
  - It picked the wrong winner (2). Even tightness probs on Stat MAX of ATs on node3 picks wrong winner (2).
- Also, no continuity in projected slack due to Stat MAX on ATs at node 3.
- **SPT advantages:**
  1. Chooses slack-critical inputs first at a merge point. Finds critical paths sooner.
  2. Computes correct path-slapck canonical as it walks backwards.
    - Slack at the starting node of a peeled path is precisely the projection of the path-based slack canonical computed using path-based AT along that path.
  3. Makes CPPR check against “cutoff” more meaningful.
Results

- **CPPR** (Common Path Pessimism Removal) implemented in a GBA (Graph or Block-Based Analysis) Statistical Timer
  - cutoff: Only analyze tests with original slack (OS) < specified value. Also, during path-tracing, schedule sub-critical inputs at a merge point with slack < specified value.
  - PPT (Paths Per Test): Stop further path peeling if number of paths currently peeled exceeds specified number.
  - Implemented both DPT and SPT as Path-tracing methods within CPPR.

- **Exhaustive CPPR**: unlimited cutoff and PPT.
  - Analyze all tests and peel all paths.

- **WS** (Worst Post-CPPR Slack) for a test.

- **WPI** (Worst Path Index): The path number at which WS was found for that test.

- **CPPR Safety**: $\text{PPT} \geq \max \text{WPI}$ over all tests
  - smaller WPI is better since CPPR can be run with smaller PPT; hence faster.

- **DPT vs SPT** compare on 4 industrial designs (**Exhaustive CPPR**)

<table>
<thead>
<tr>
<th>Design (# gates)</th>
<th>Total</th>
<th>Number of Tests</th>
<th>Max WPI-diff = 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total</td>
<td>WPI$<em>{SPT} &gt;$ WPI$</em>{DPT}$</td>
<td>WPI$<em>{SPT} &lt;$ WPI$</em>{DPT}$</td>
</tr>
<tr>
<td>D1 (13K)</td>
<td>30K</td>
<td>12</td>
<td>2K</td>
</tr>
<tr>
<td>D2 (0.5K)</td>
<td>2K</td>
<td>0</td>
<td>0.1K</td>
</tr>
<tr>
<td>D3 (353K)</td>
<td>806K</td>
<td>451</td>
<td>64K</td>
</tr>
<tr>
<td>D4 (12K)</td>
<td>25K</td>
<td>22</td>
<td>2K</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Design</th>
<th>Max WPI$_{DPT}$</th>
<th>Max WPI$_{SPT}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>305</td>
<td>5</td>
</tr>
<tr>
<td>D2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>D3</td>
<td>8364</td>
<td>31</td>
</tr>
<tr>
<td>D4</td>
<td>114</td>
<td>9</td>
</tr>
</tbody>
</table>

SPT is clearly superior
Results (Contd.)

WPI Compare on D1

- SPT is clearly superior
- Max $WPI_{SPT} = 5$
- Max $WPI_{DPT} = 305$

Slack Compare on a Setup Test in D3

- DPT_Pre is monotonic
- But no relation to DPT_Post (wild)
- DPT needs 508 paths to find WS
- while SPT finds the same WS on path #1

Slack-Delta = Path-Slack - OS

WPI = 1
99.75% (SPT) 93.12% (DPT)

SPT_Pre is not-monotonic
But tracks SPT_Post

Common Credit

WS-Delta = -14.058

$WPI_{SPT} = 1$ $WPI_{DPT} = 508$
Final Thoughts

1. SPT finds post-cppr critical paths sooner than DPT
   - SPT has smaller WPI
   - Can run CPPR with much smaller PPT; hence faster.

2. Running CPPR with cutoff = 10 on design D3
   - 99,889 tests were analyzed
   - 124 (i.e. 0.124%) of these showed $WS_{DPT} > WS_{SPT}$; so CPPR with DPT is unsafe when run with a cutoff.
     a. 9 of these were Setup Tests with diff $WS_{DPT} - WS_{SPT} \in [13.038, 33.711]$ ps
     b. 115 of these were Hold Tests with diff $WS_{DPT} - WS_{SPT} \in [0.0037, 13.083]$ ps
   - DPT in all except 15 out of 124 of these tests peeled only 1 path because all sub-critical branches weren’t scheduled for future peeling since their “pathSlack” exceeded the given cutoff value.

3. In Exhaustive CPPR, SPT is about 2X slower than DPT
   - Not surprising considering the extra stat calculations needed in SPT vs DPT

4. In a “Production” CPPR run on D3, SPT has been seen to be 30% faster than DPT due to much smaller PPT budget.
   - Much fewer paths peeled by SPT due to much smaller WPI.