GPU-Accelerated Static Timing Analysis

Zizheng Guo¹, Tsung-Wei Huang², Yibo Lin¹

¹CS Department, Peking University
²ECE Department, University of Utah
Outline

- Introduction
  - Static timing analysis (STA)
  - Previous work on STA acceleration

- Problem formulation and our proposed algorithms
  - RC delay computation
  - Levelization
  - Timing propagation

- Experimental result

- Conclusion
Static Timing Analysis: Basic Concepts

- Correct functionality
- Performance

Image source:
https://www.synopsys.com/glossary/what-is-static-timing-analysis.html
https://sites.google.com/site/taucontest2015/
Static Timing Analysis: Basic Concepts

- Correct functionality and performance
- Simplified delay models
  - Cell delay: non-linear delay model (NLDM)
  - Net delay: Elmore delay model (Parasitic RC Tree)
Static Timing Analysis: Call For Acceleration

- Time-consuming for million/billion-size VLSI designs
- Need to be called many times to guide optimization
  - Timing-driven placement, timing-driven routing etc.

Image source: ePlace [Lu, TODAES’15], Dr. CU [Chen, TCAD’20]
Prior Works and Challenges

- Parallelization on CPU by multithreading
  - [Huang, ICCAD’15] [Lee, ASP-DAC’18]...
  - cannot scale beyond 8-16 threads

- Statistical STA acceleration using GPU
  - [Gulati, ASPDAC’09] [Cong, FPGA’10]...
  - Less challenging than conventional STA

Image source:
[Huang, TCAD’20]
Prior Works and Challenges

- Accelerate STA using modern GPU
  - Lookup table query and timing propagation [Wang, ICPP’14] [Murray, FPT’18]
  - 6.2x kernel time speed-up, but 0.9x of entire time because of data copying

- Leveraging GPU is challenging
  - Graph-oriented: diverse computational patterns and irregular memory access
  - Data copy overhead

Image source:
[Huang, TCAD’20]
Fully GPU-Accelerated STA

- Efficient GPU algorithms
  - Covers the runtime bottlenecks
- Implementation based on open source STA engine OpenTimer

https://github.com/OpenTimer/OpenTimer
RC Delay Computation

- The Elmore delay model explained.

- \( \text{load}_u = \sum_v \text{is child of } u \ \text{cap}_v \)
  - eg. \( \text{load}_A = \text{cap}_A + \text{cap}_B + \text{cap}_C + \text{cap}_D = \text{cap}_A + \text{load}_B + \text{load}_D \)

- \( \text{delay}_u = \sum_v \text{any node } \text{cap}_v \times R_{Z\rightarrow LCA(u,v)} \)
  - eg. \( \text{delay}_B = \text{cap}_A R_{Z\rightarrow A} + \text{cap}_D R_{Z\rightarrow A} + \text{cap}_B R_{Z\rightarrow B} + \text{cap}_C R_{Z\rightarrow B} = \text{delay}_A + R_{A\rightarrow B} \text{load}_B \)
RC Delay Computation

- The Elmore delay model explained.

\[ l_{\text{delay}}_u = \sum_{v \text{ is child of } u} \text{cap}_v \times \text{delay}_v \]

- \[ \beta_v = \sum_{v \text{ is any node}} \text{cap}_v \times \text{delay}_v \times R_{Z \rightarrow LCA(u,v)} \]
RC Delay Computation

- Flatten the RC trees by parallel BFS and counting sort on GPU.
- Store only parent index of each node on GPU
- Redesign the dynamic programming on trees

Parent list representation in memory

- Z
- Z
- A
- A
- B
RC Delay Computation

- Store only parent index of each node on GPU
- Redesign the dynamic programming on trees

**DFS_load(u):**

\[
\text{load}[u] = \text{cap}[u] \\
\text{For child } v \text{ of } u:\ \\
\text{DFS_load}(v) \\
\text{load}[u] += \text{load}[v]
\]

**GPU_load:**

For \( u \) in [C, D, B, E, A]:

\[
\text{load}[u] += \text{cap}[u] \\
\text{load}[u\text{.parent}] += \text{load}[u]
\]
RC Delay Computation

- Store only parent index of each node on GPU, and re-implement the dynamic programming on trees, based on the direction of value update.

DFS\_delay(u):
For child v of u:
\[\text{temp} := R[u,v] \times \text{load}[v]\]
\[\text{delay}[v] = \text{delay}[u] + \text{temp}\]
\[\text{DFS\_delay}(v)\]

GPU\_delay:
For u in \([A, E, B, D, C]\):
\[\text{temp} := R[u\.parent,u] \times \text{load}[u]\]
\[\text{delay}[u]=\text{delay}[u\.parent] + \text{temp}\]
Global memory read/write introduces delay. GPU will automatically coalesce adjacent memory requests.
Task Graph Levelization

- Build level-by-level dependencies for timing propagation tasks.
  - Essentially a parallel topological sorting.
- Maintain a set of nodes called frontiers, and update the set using “advance” operation.
Task Graph Levelization: Reverse Technique

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>#nodes</th>
<th>Max In-degree</th>
<th>Max Out-degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>netcard</td>
<td>3999174</td>
<td>8</td>
<td>260</td>
</tr>
<tr>
<td>vga_lcd</td>
<td>397809</td>
<td>12</td>
<td>329</td>
</tr>
<tr>
<td>wb_dma</td>
<td>13125</td>
<td>12</td>
<td>95</td>
</tr>
</tbody>
</table>
GPU Look-up Table Query

- Do linear interpolation/extrapolation and eliminate unnecessary branches
  - Unified inter-/extrapolation
  - Degenerated LUTs
Experiment Setup

- Nvidia CUDA, RTX 2080, 40 Intel Xeon Gold 6138 CPU cores
- RC Tree Flattening
  - 64 threads per block with one block for each net
- Elmore delay computation
  - 4 threads for each net (one for each Early/Late and Rise/Fall condition) with a block of 64 nets
- Levelization
  - 128 threads per block
- Timing propagation
  - 4 threads for each arc, with a block of 32 arcs
Experimental Results

- Up to $3.69 \times$ speed-up (including data copy)
- Bigger performance margin with bigger problem size

Fig. 9: Runtime breakdown of the circuit leon2 (21M nodes).
Experimental Results

- Up to $3.69 \times$ speed-up (including data copy)
- Bigger performance margin with bigger problem size

Fig. 11: Runtime values at different numbers of CPUs. Our runtime under 1 CPU and 1 GPU is close to OpenTimer of 40 CPUs.
Experimental Results (Incremental Timing)

- Break-even point
  - 45K nets and gates
  - 67K propagation candidates
- Useful for timing driven optimization
- Mixed strategy
Conclusions and Future Work

- Conclusions:
  - GPU-accelerated STA that go beyond the scalability of existing methods
  - GPU-efficient data structures and algorithms for delay computation, levelization and timing propagation
  - Up to 3.69x speedup

- Future Work
  - Explore different cell/net delay models.
  - Develop efficient GPU algorithms for CPPR
Thanks!

Questions are welcome

Website: https://guozz.cn
Email: gzz@pku.edu.cn