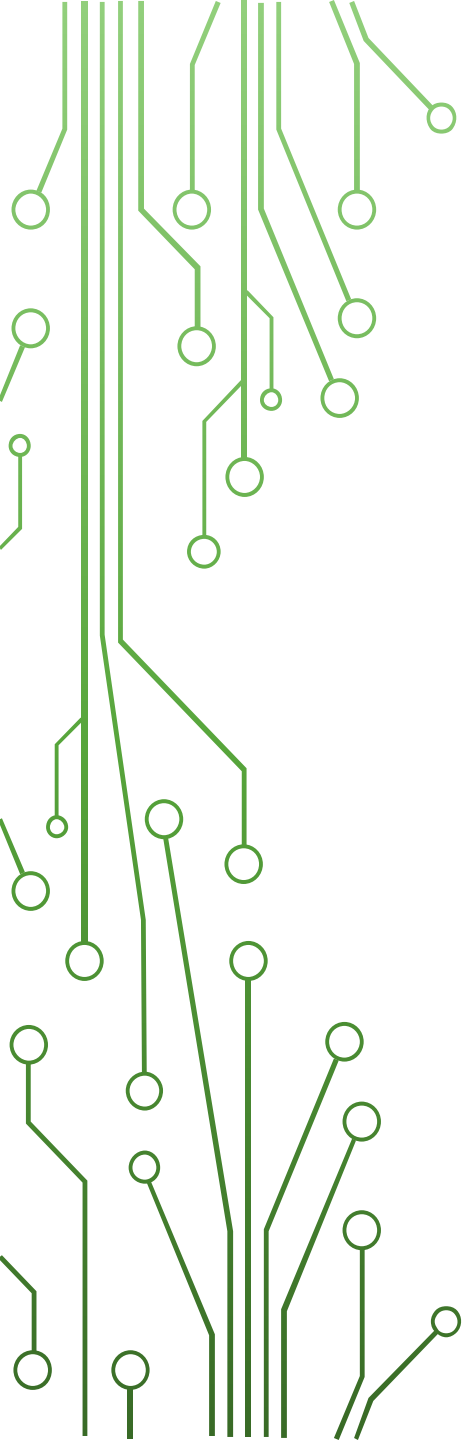


北京大学高能效计算与应用中心
Center for Energy-efficient Computing and Applications

GPU-Accelerated Static Timing Analysis

Zizheng Guo[1], Tsung-Wei Huang[2], Yibo Lin[1]

[1]CS Department, Peking University

[2]ECE 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

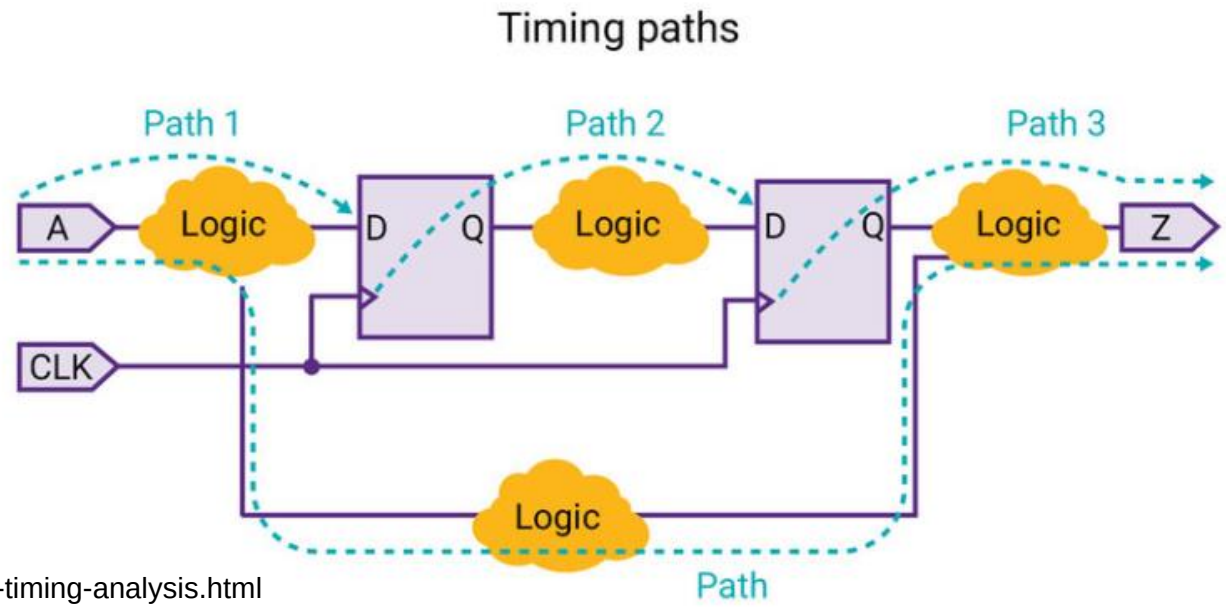
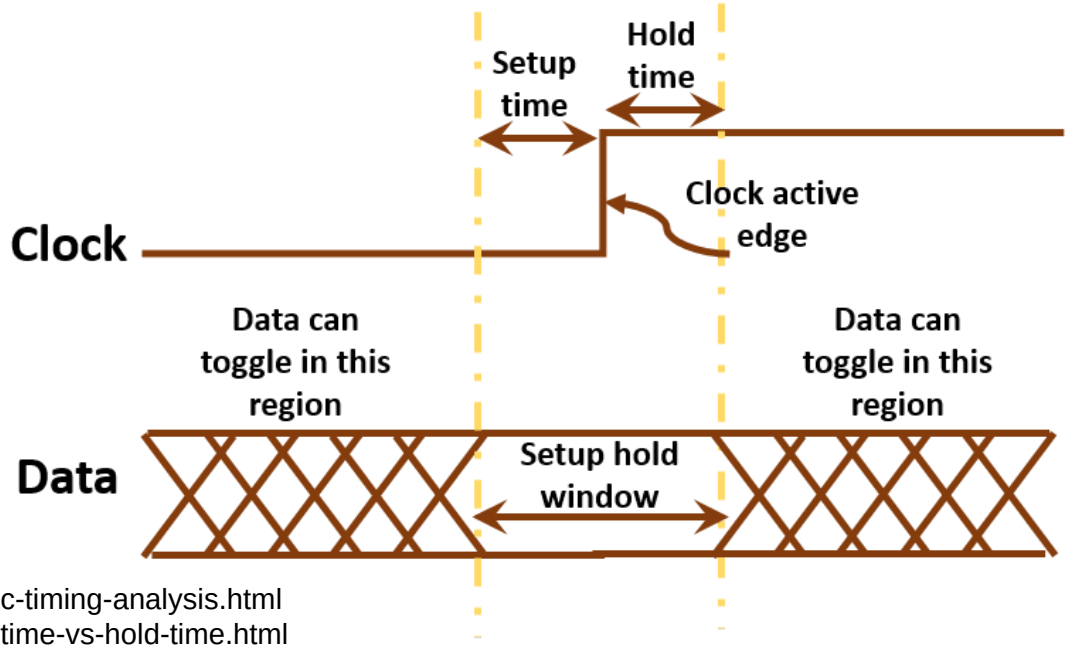


Image source:
https://www.synopsys.com/glossary/what-is-static-timing-analysis.html
https://vlsiuniverse.blogspot.com/2016/12/setup-time-vs-hold-time.html
https://sites.google.com/site/taucontest2015/

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)



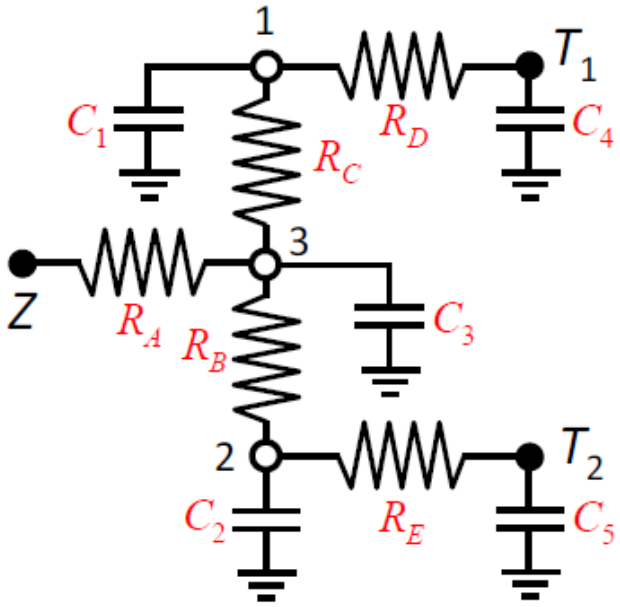
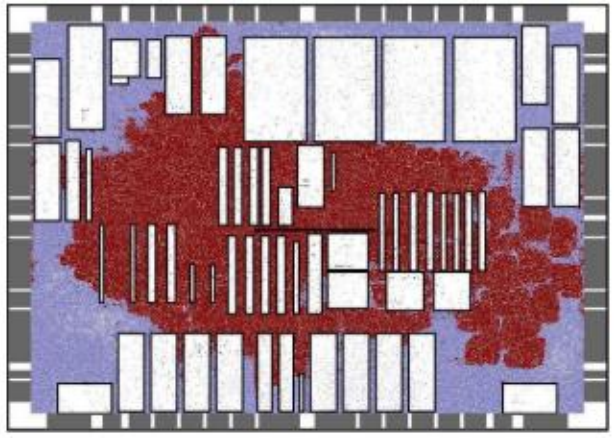


Image source:
https://www.synopsys.com/glossary/what-is-static-timing-analysis.html
https://vlsiuniverse.blogspot.com/2016/12/setup-time-vs-hold-time.html
https://sites.google.com/site/taucontest2015/

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.



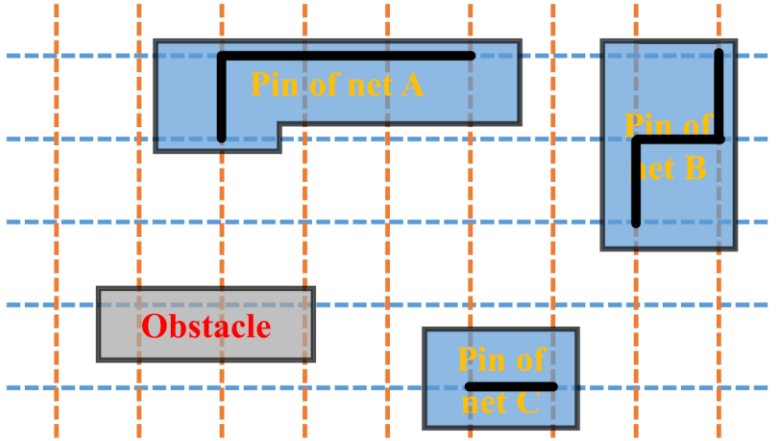
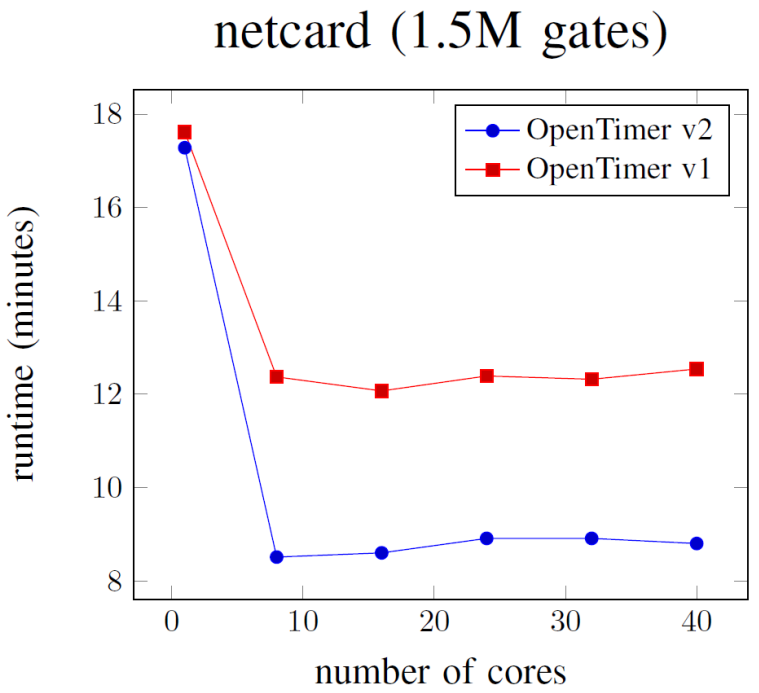


Image source: ePlace [Lu, TODAES'15], Dr. CU [Chen, TCAD'20]

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



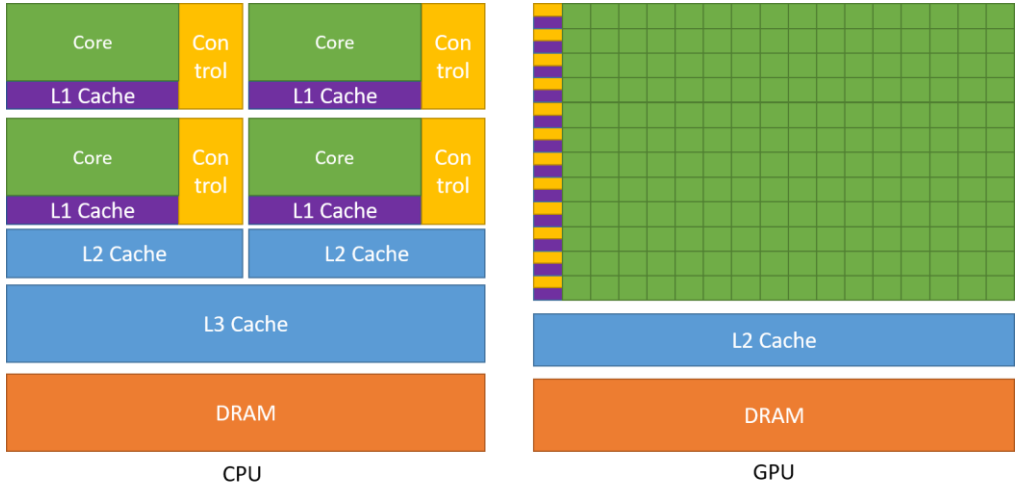
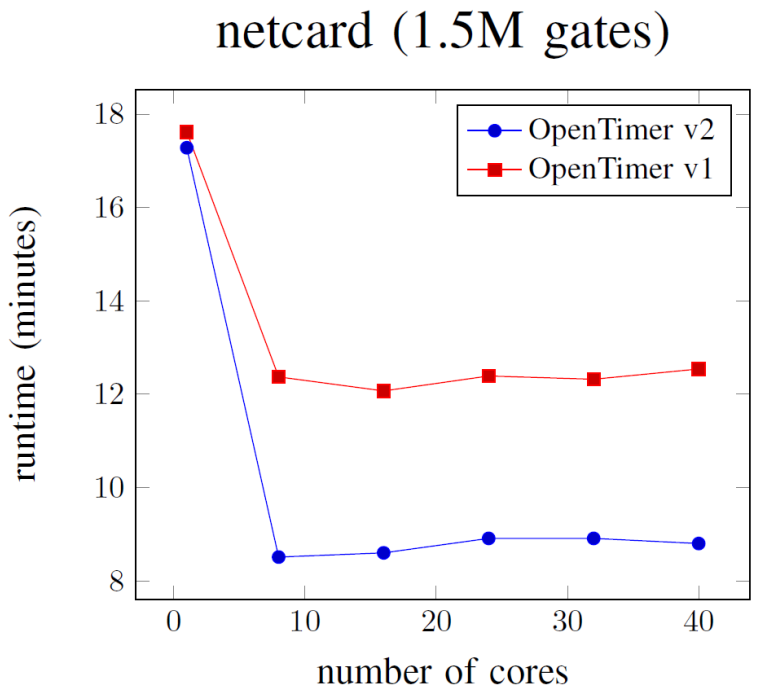


Image source:
[Huang, TCAD'20]
https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#introduction

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



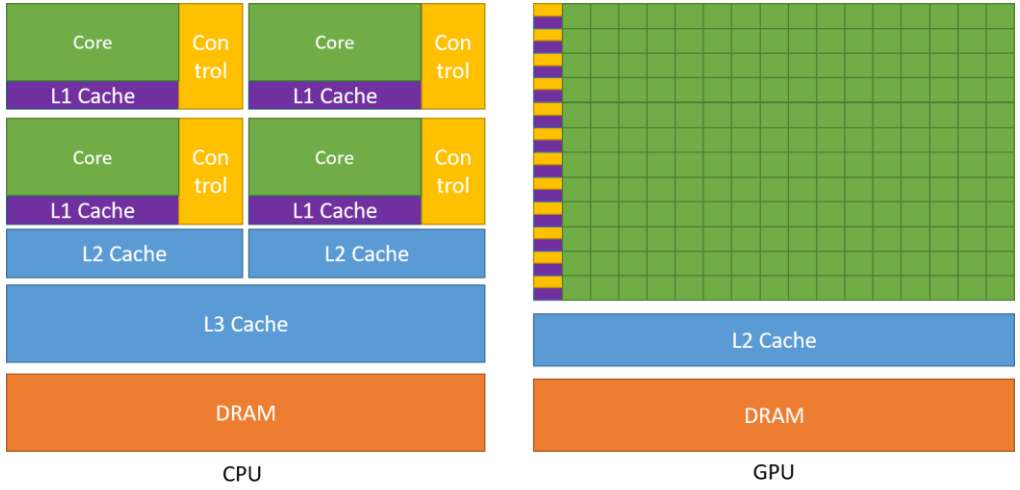
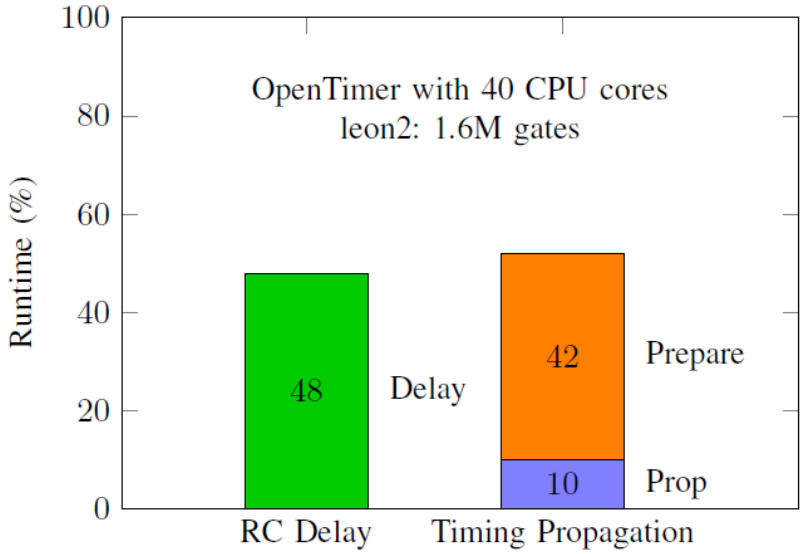
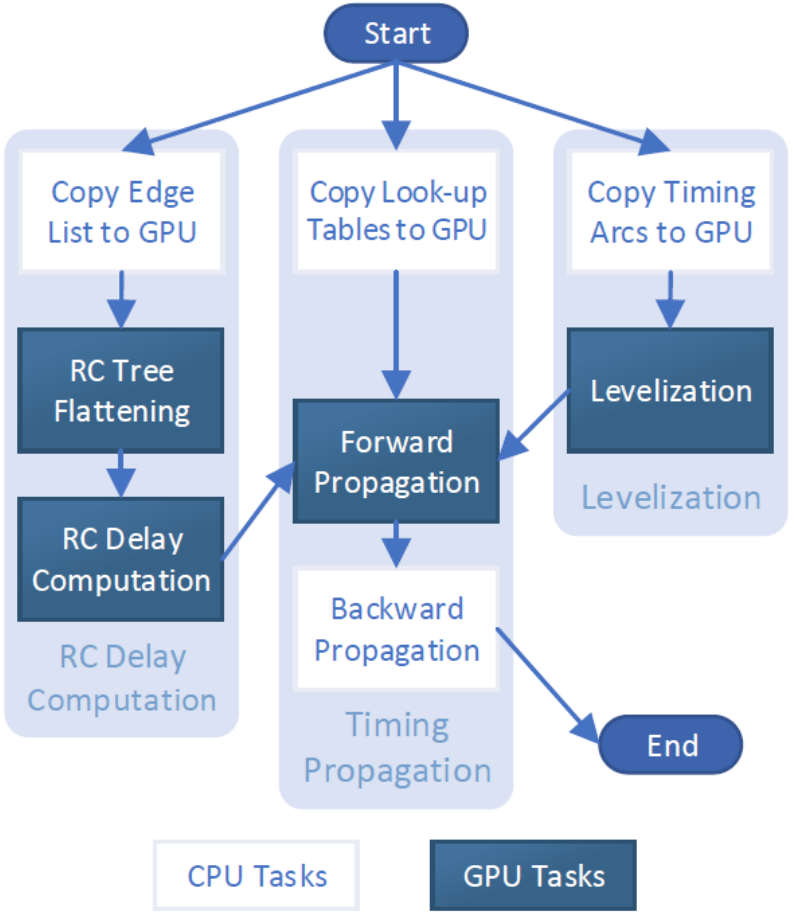


Image source:
[Huang, TCAD'20]
https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#introduction

Fully GPU-Accelerated STA

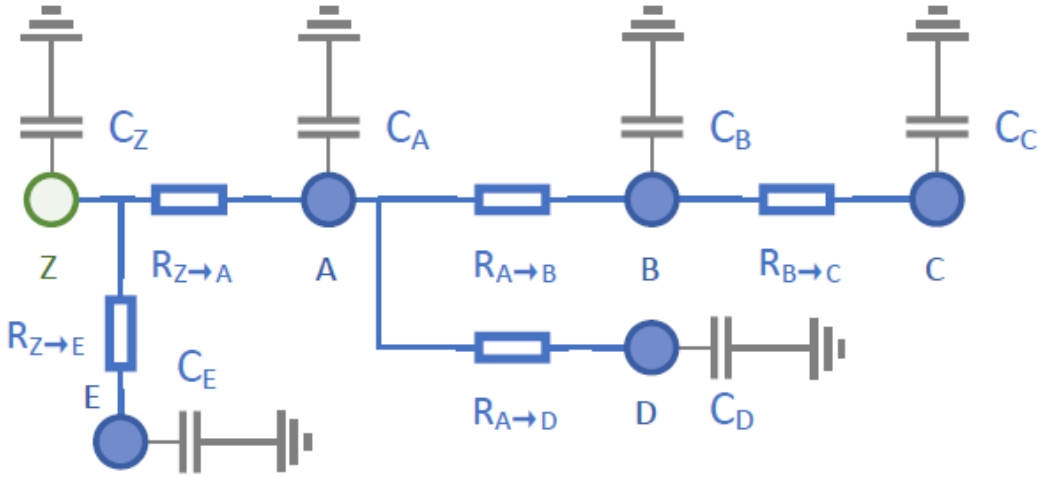
- Efficient GPU algorithms
  - Covers the runtime bottlenecks
- Implementation based on open source STA engine OpenTimer



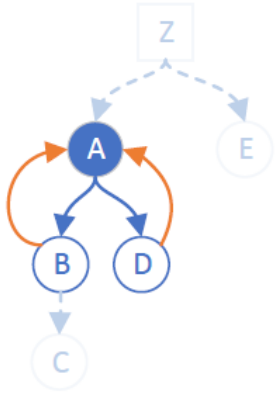


https://github.com/OpenTimer/OpenTimer

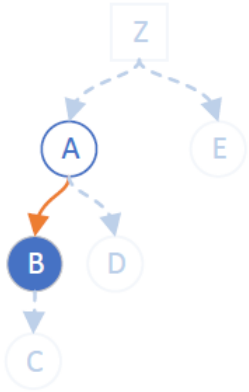
RC Delay Computation



- The Elmore delay model explained.
- $load_u = \sum_{v \text{ is child of } u} cap_v$
  - eg. $load_A = cap_A + cap_B + cap_C + cap_D = cap_A + load_B + load_D$
- $delay_u = \sum_{v \text{ is any node}} cap_v \times R_{Z\to LCA(u,v)}$
  - eg. $delay_B = cap_A R_{Z\to A} + cap_D R_{Z\to A} + cap_B R_{Z\to B} + cap_C R_{Z\to B} = delay_A + R_{A\to B} load_B$



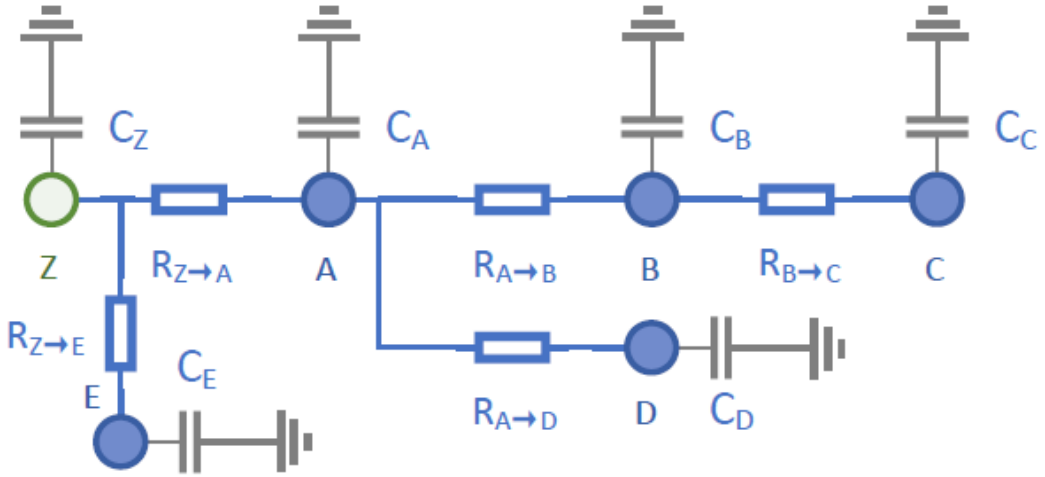
(a) Upward

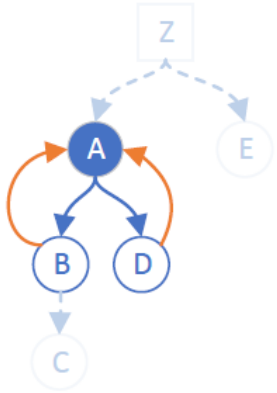


(b) Downward

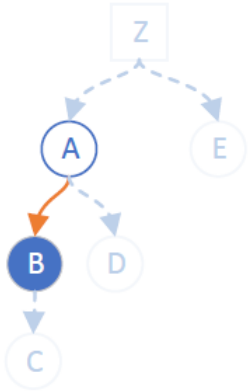
RC Delay Computation

- The Elmore delay model explained.
- $ldelay_u = \sum_{v \text{ is child of } u} cap_v \times delay_v$
- $\beta_v = \sum_{v \text{ is any node}} cap_v \times delay_v \times R_{Z \to LCA(u,v)}$





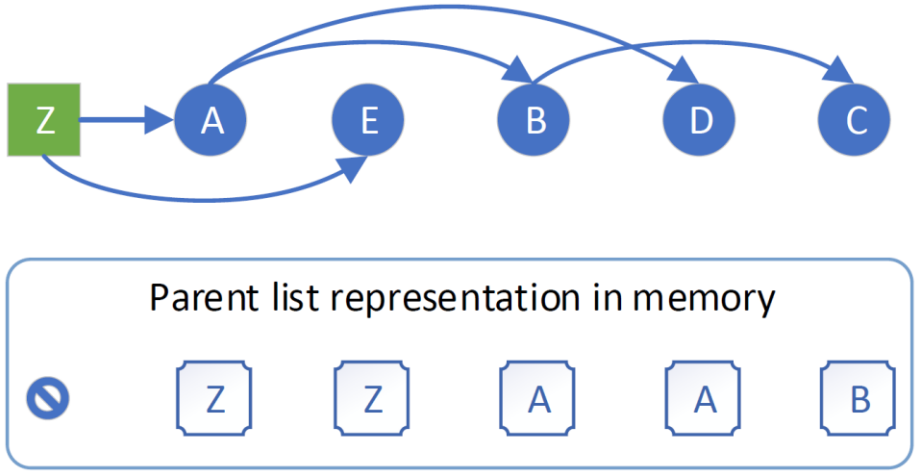
(a) Upward



(b) Downward

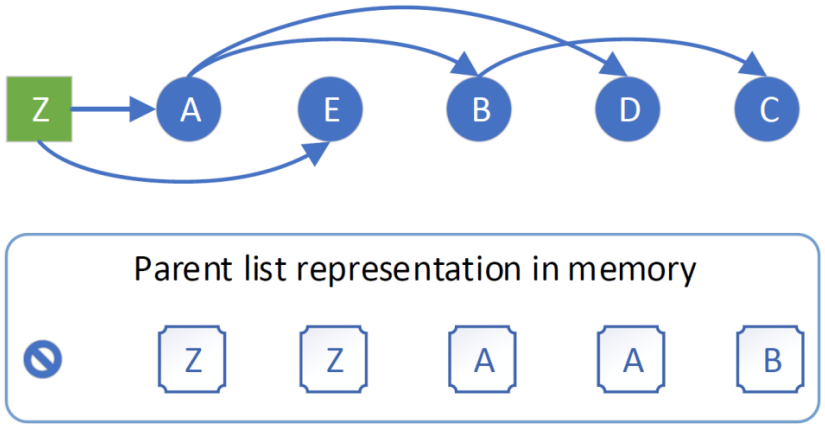
RC Delay Computation

- 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



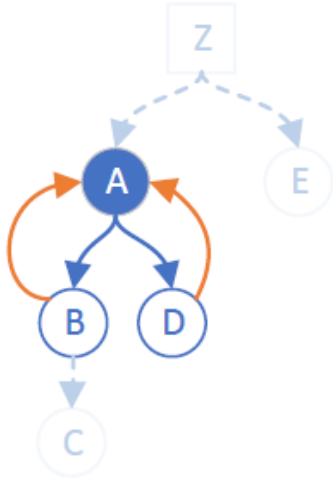
RC Delay Computation

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




```
DFS_load(u):
    load[u] = cap[u]
    For child v of u:
        DFS_load(v)
        load[u] += load[v]
```

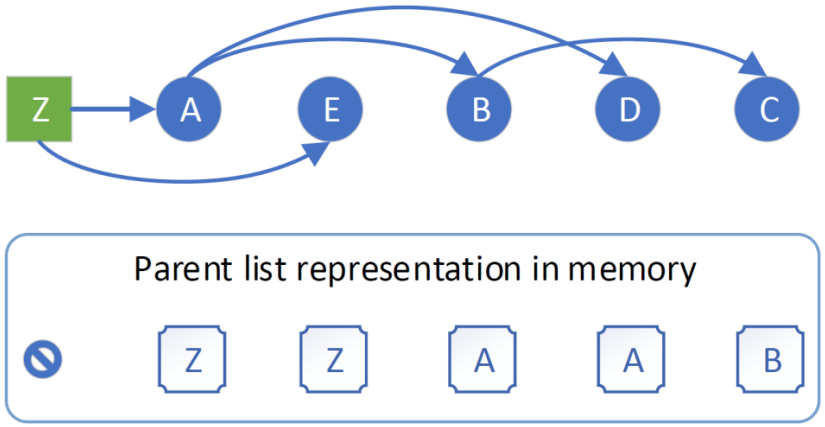
```
GPU_load:
    For u in [C, D, B, E, A]:
        load[u] += cap[u]
        load[u.parent] += load[u]
```



(a) Upward

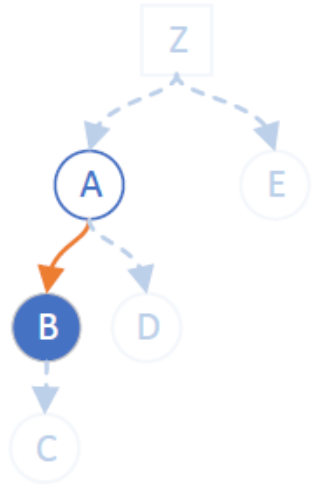
RC Delay Computation

- 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:
        temp := R[u,v]*load[v]
        delay[v] = delay[u] + temp
        DFS_delay(v)
```

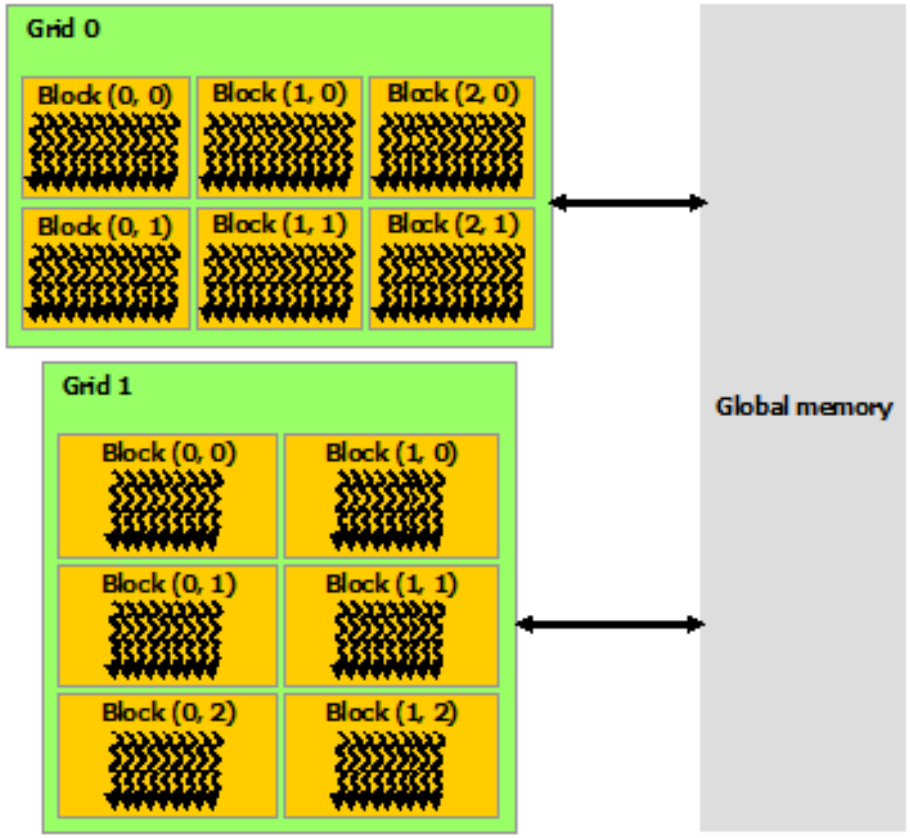
```
GPU_delay:
    For u in [A, E, B, D, C]:
        temp := R[u.parent,u]*load[u]
        delay[u]=delay[u.parent] + temp
```



(b) Downward

RC Delay Memory Coalesce

- Global memory read/write introduces delay. GPU will automatically coalesce adjacent memory requests.



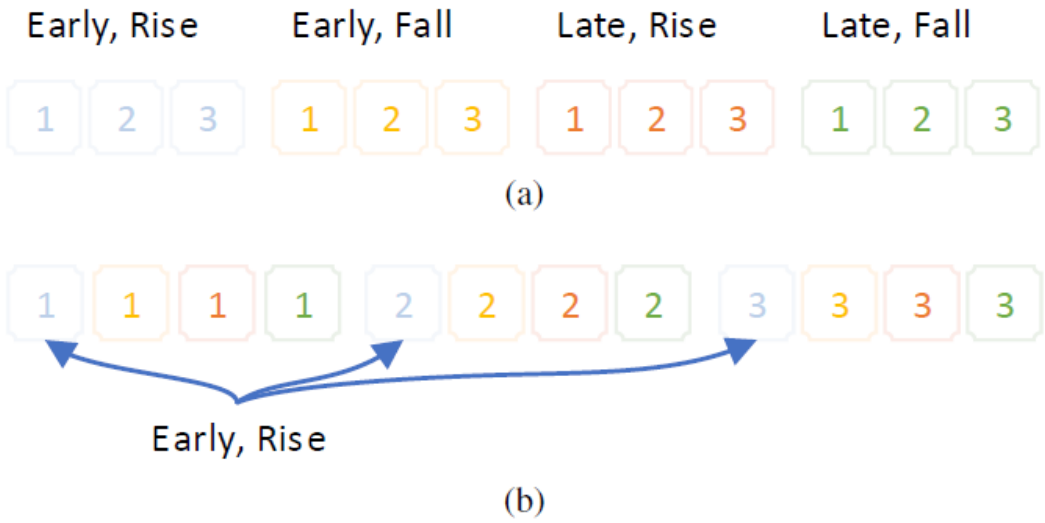
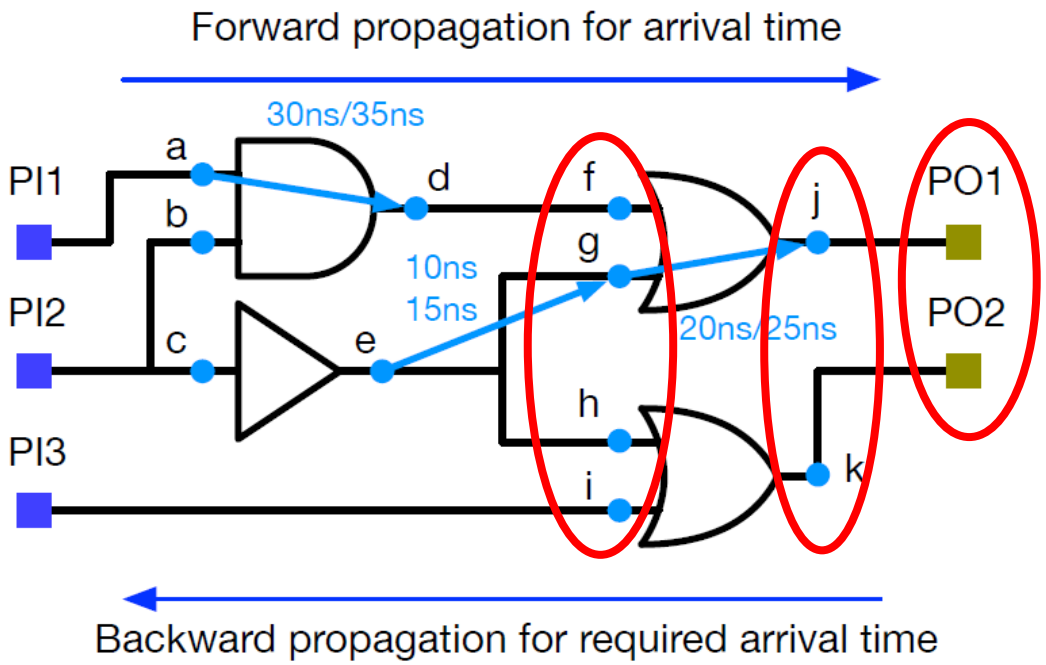
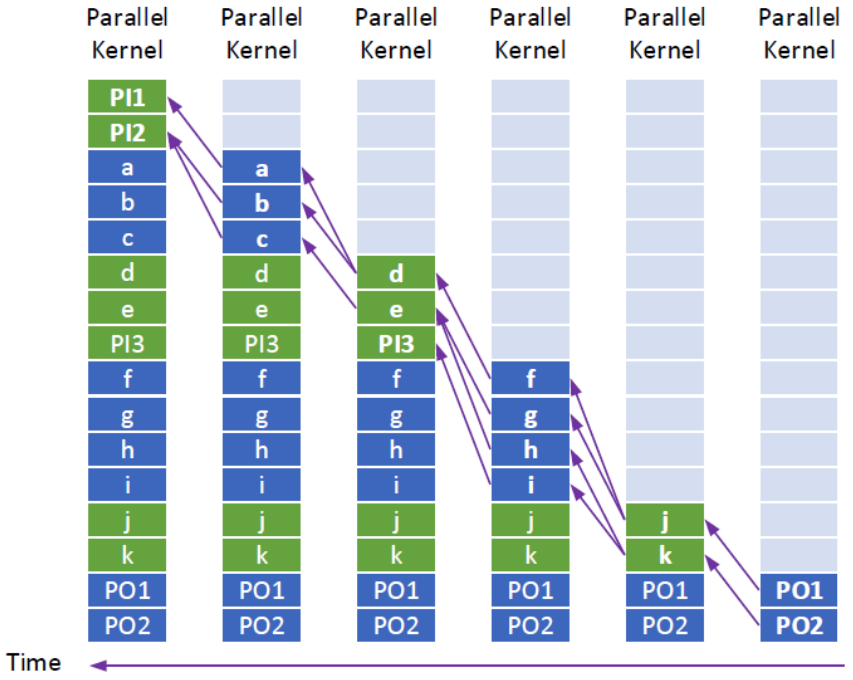


Image source: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#memory-hierarchy

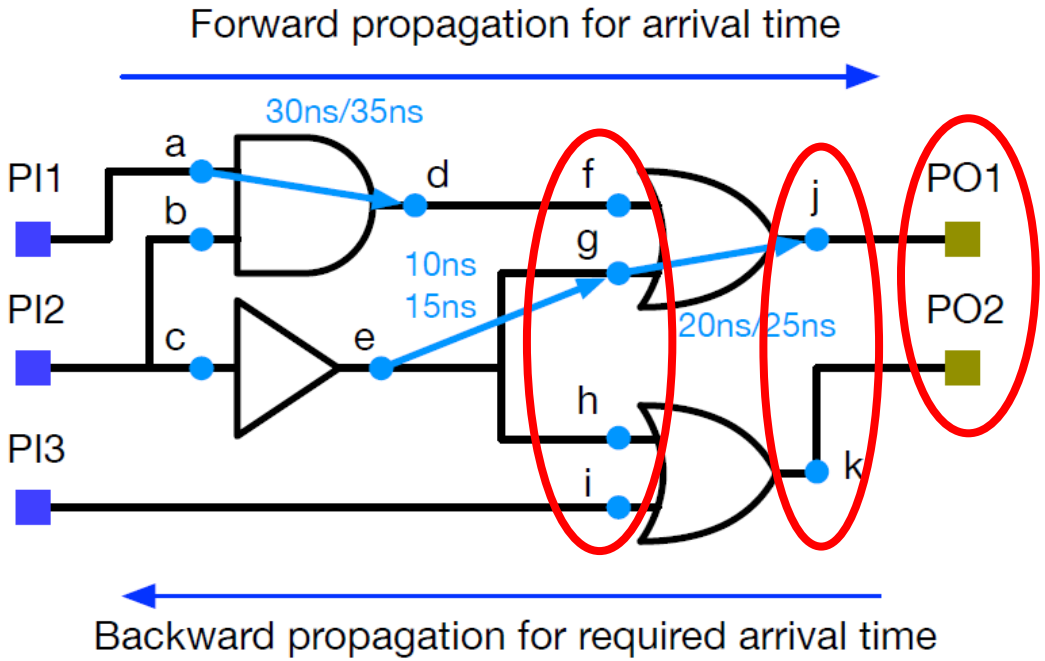
Task Graph Levelization

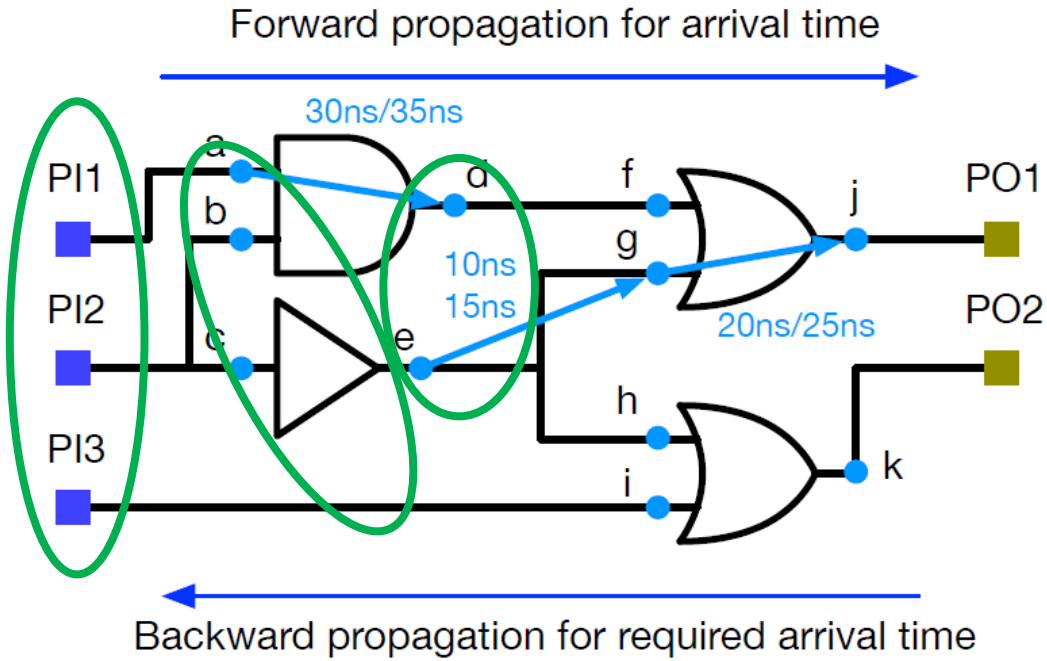
- Build level-by-level dependencies for timing propagation tasks.
  - Essentially a parallel topological sorting.
- Maintain a set of nodes called frontiers, and update the set using “advance” operation.





Task Graph Levelization: Reverse Technique

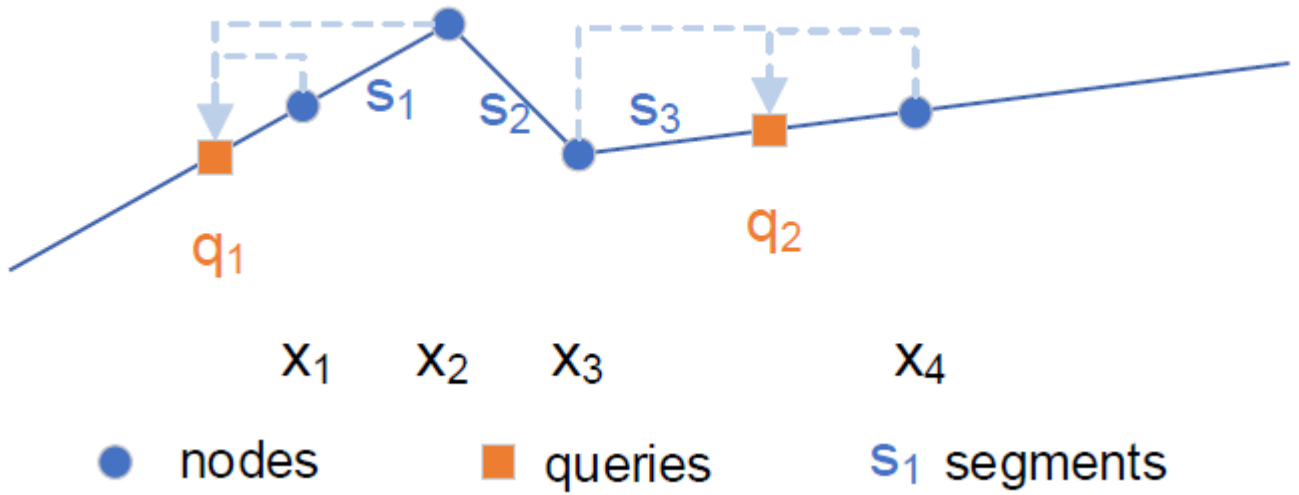




Benchmark	#nodes	Max In-degree	Max Out-degree
netcard	3999174	8	260
vga_lcd	397809	12	329
wb_dma	13125	12	95

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× speed-up (including data copy)
- Bigger performance margin with bigger problem size

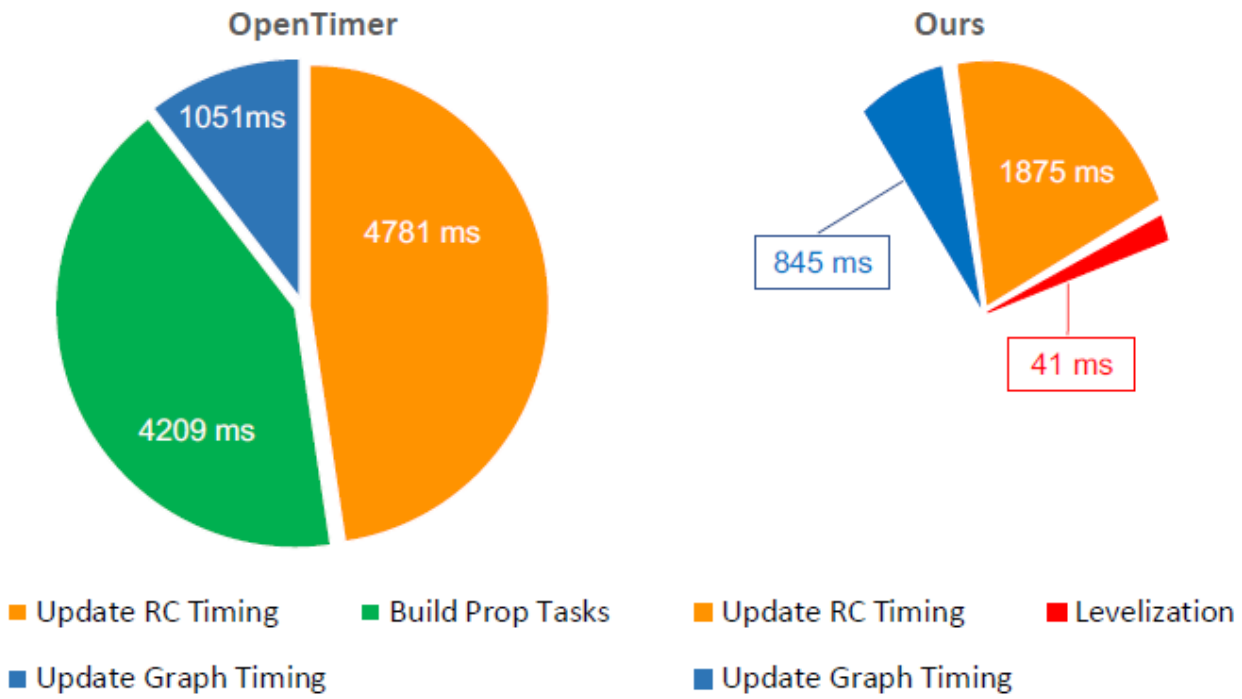


Fig. 9: Runtime breakdown of the circuit leon2 (21M nodes).

Experimental Results

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

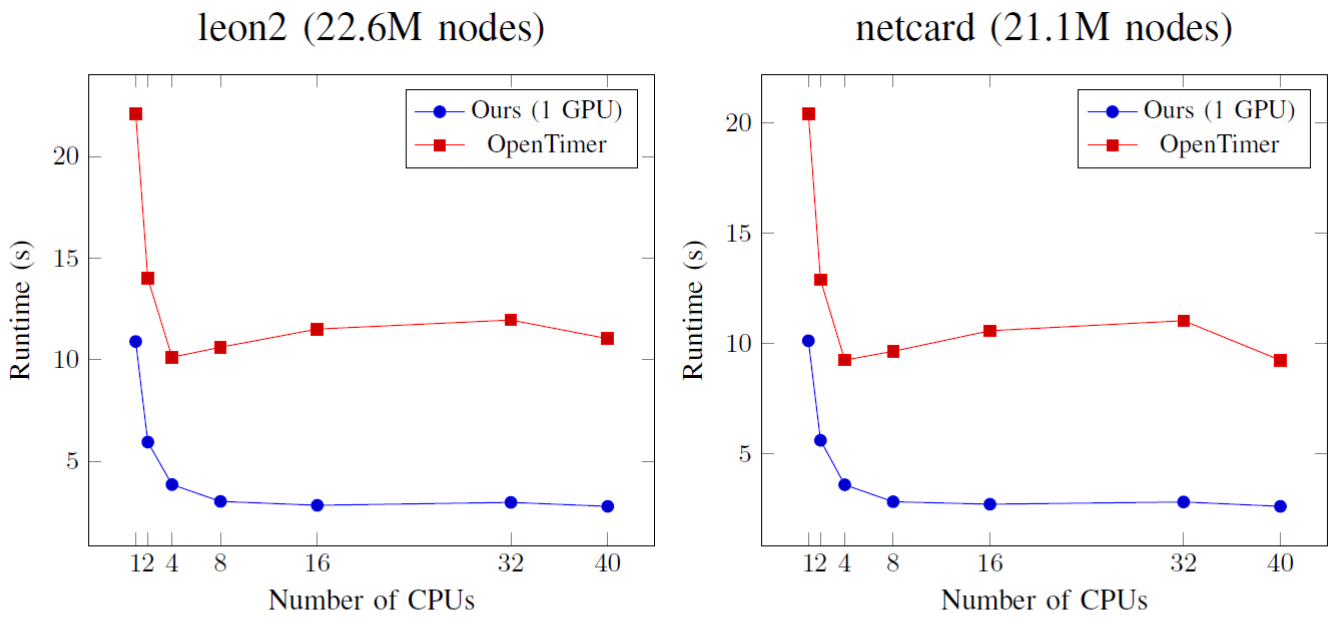
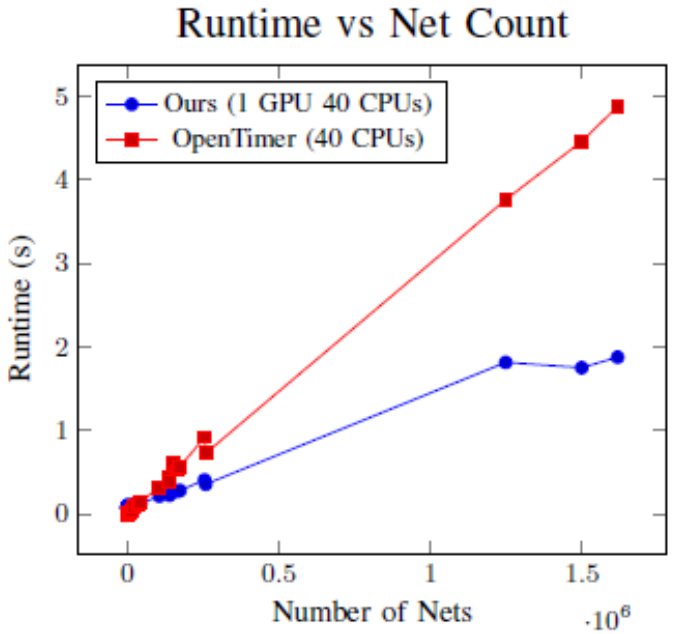
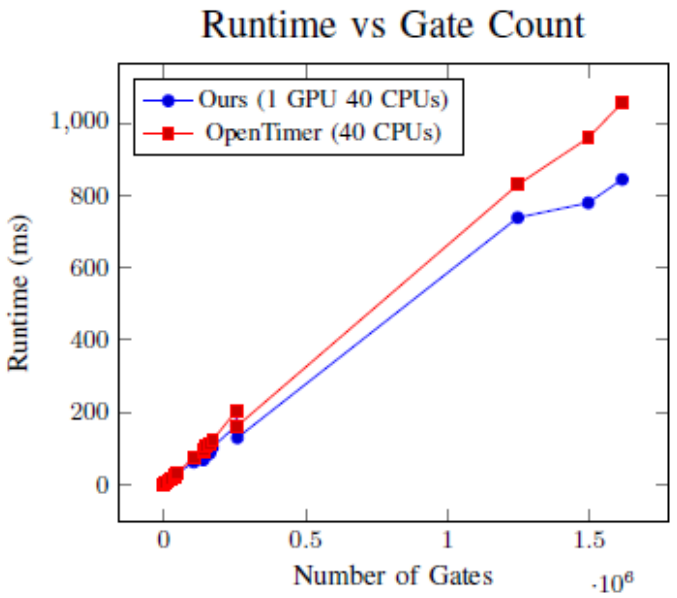


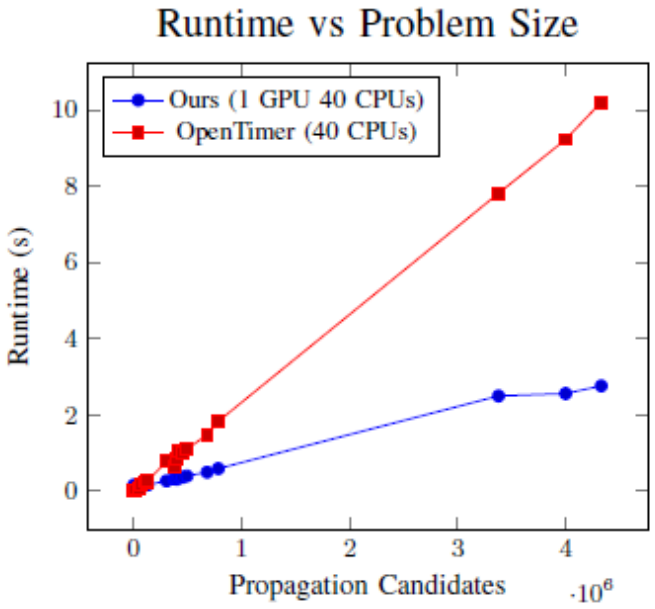
Fig. 11: Runtime values at different numbers of CPUs. Our runtime under 1 CPU and 1 GPU is close to OpenTimer of 40 CPUs.

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

北京大学高能效计算与应用中心
Center for Energy-efficient Computing and Applications

Thanks!
Questions are welcome

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