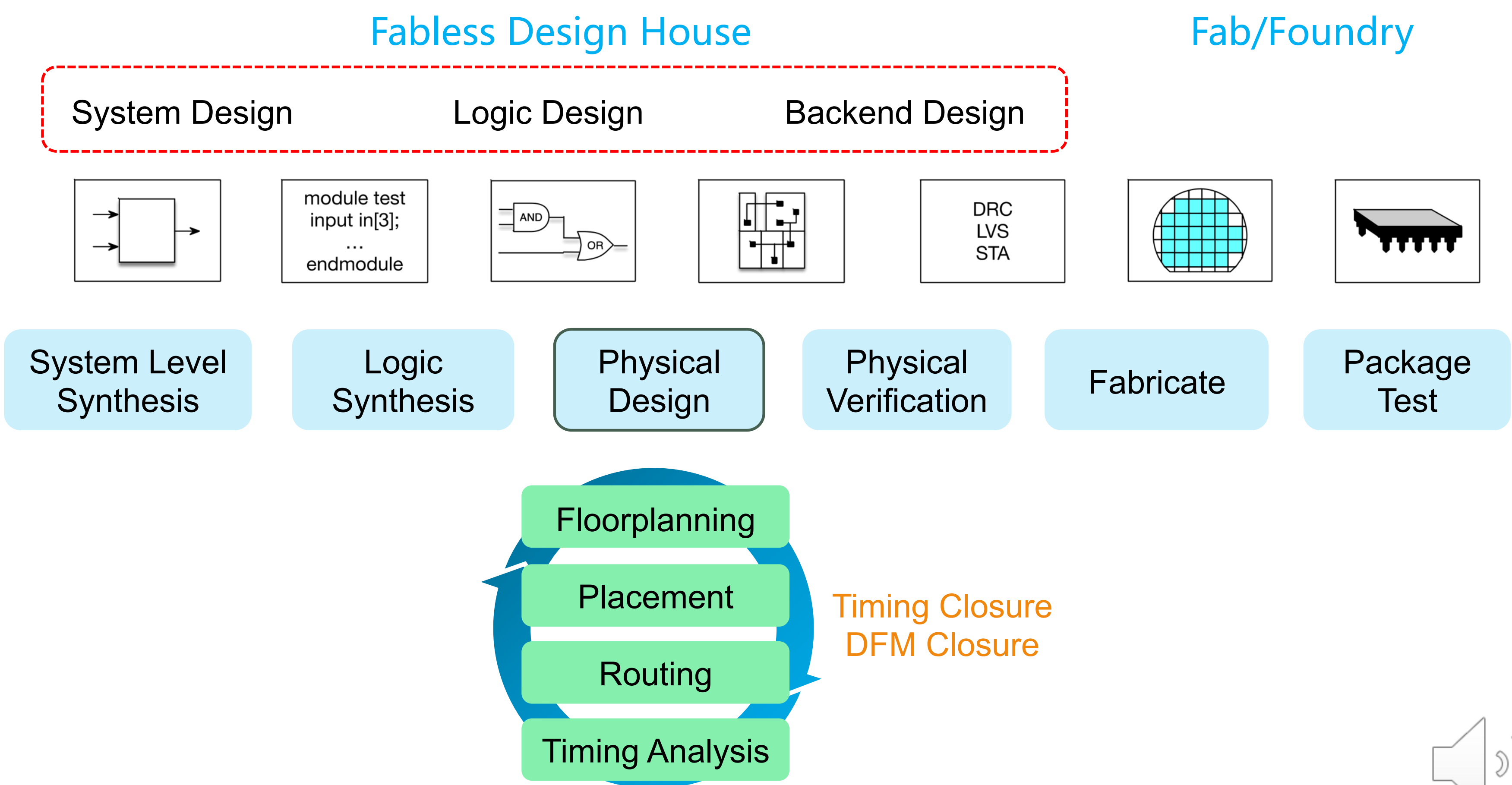
Machine Learning System-Enabled GPU Acceleration for EDA

Dr. Tsung-Wei Huang
Department of Electrical and Computer Engineering,
University of Utah, UT

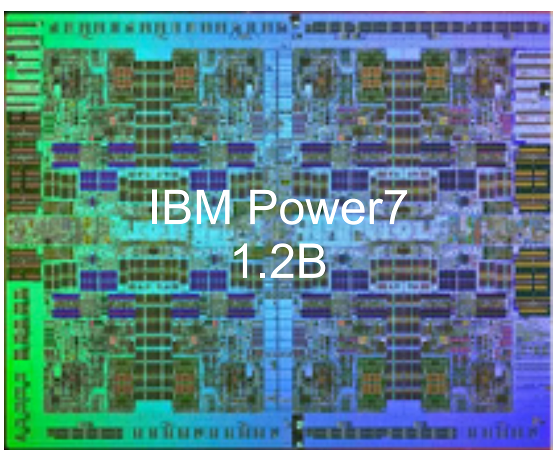
Integrated Circuits (ICs) Design Flow

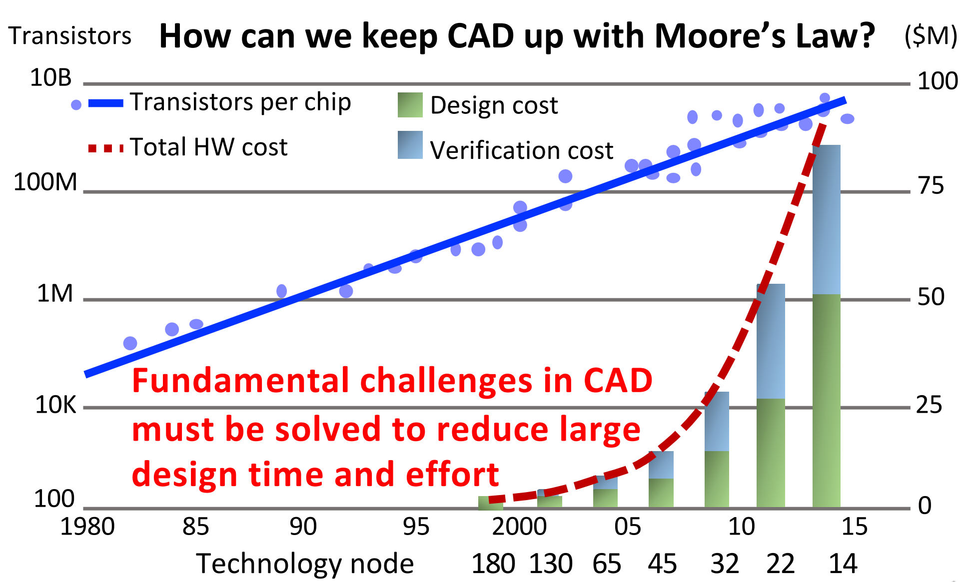
- ❑ **Electronic design automation (EDA) is a key step**



EDA is an Extremely Challenging Step

- **Large scale: billions of transistors**
- **Numerous constraints from low-level manufacturing & high-level architecture**
- **Complicated design flow**
- **Long design cycles**

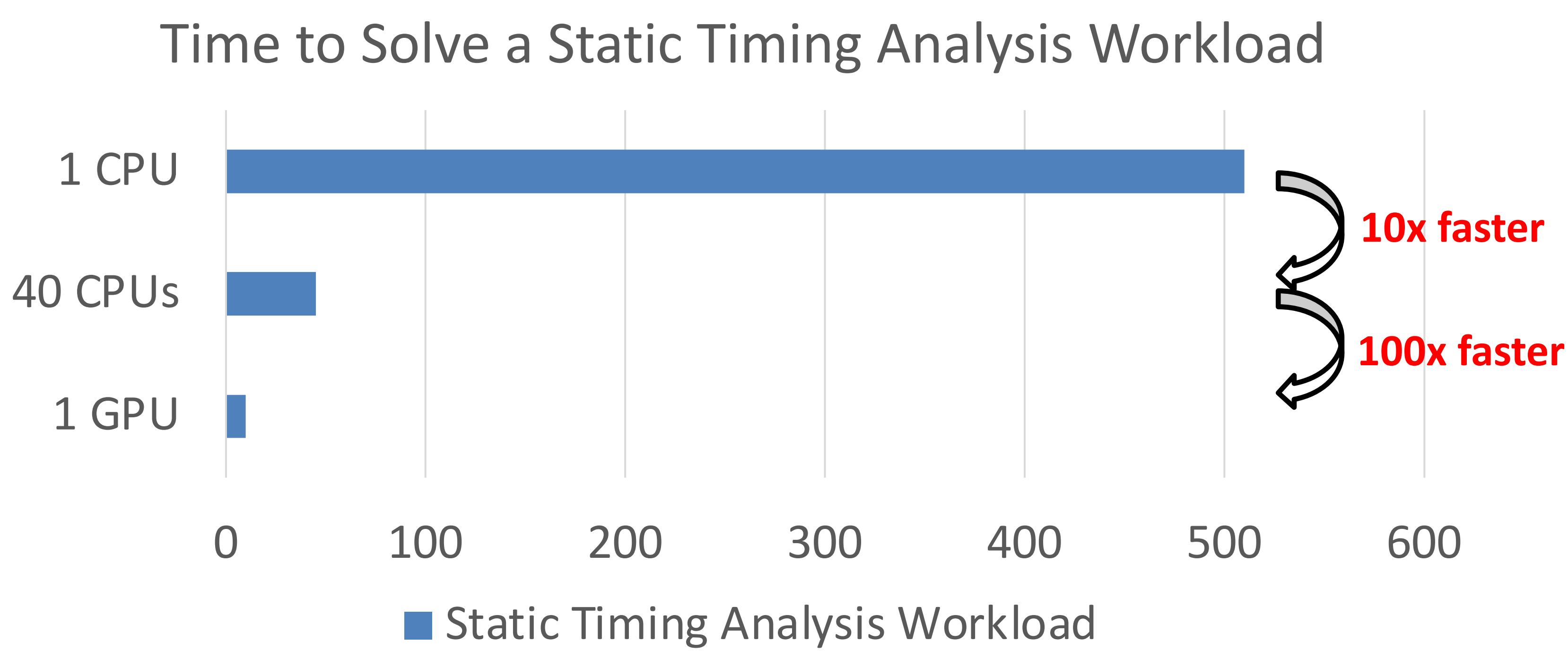




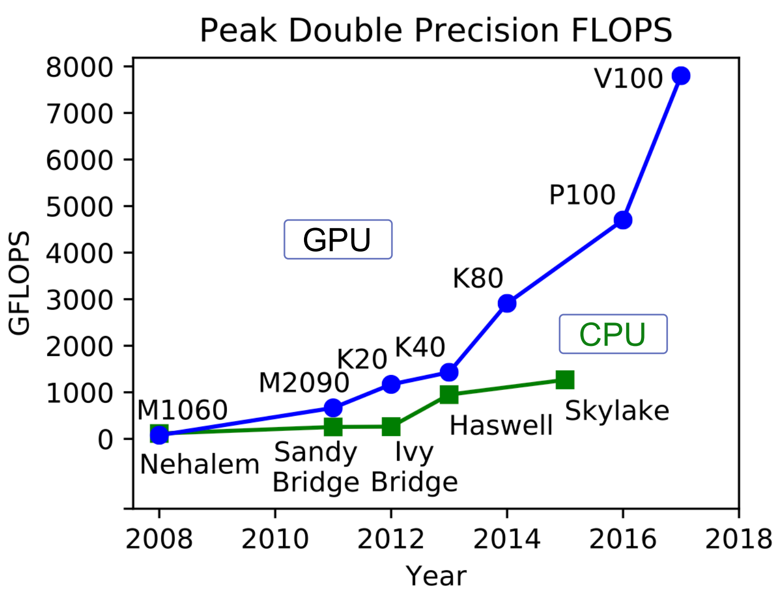
Source: DARPA IDEA Program

EDA Must Incorporate New Parallelism

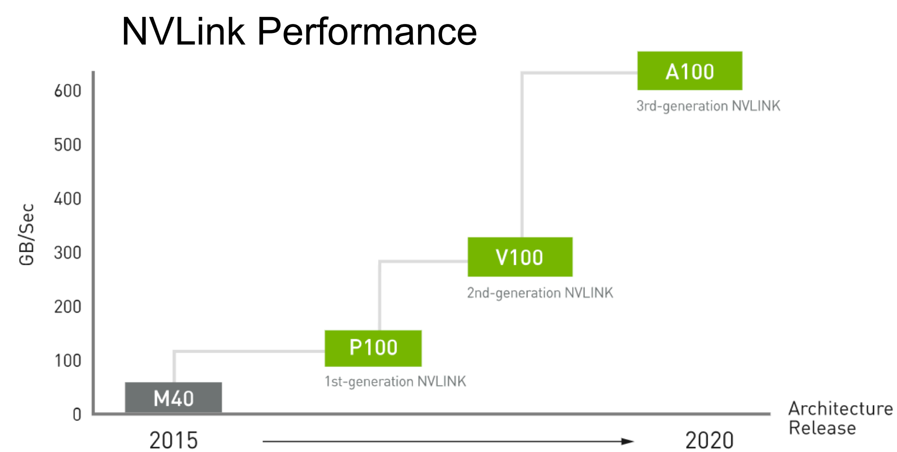
- ❑ **Manycore central processing units (CPUs)**
- ❑ **Graphics processing units (GPUs)**

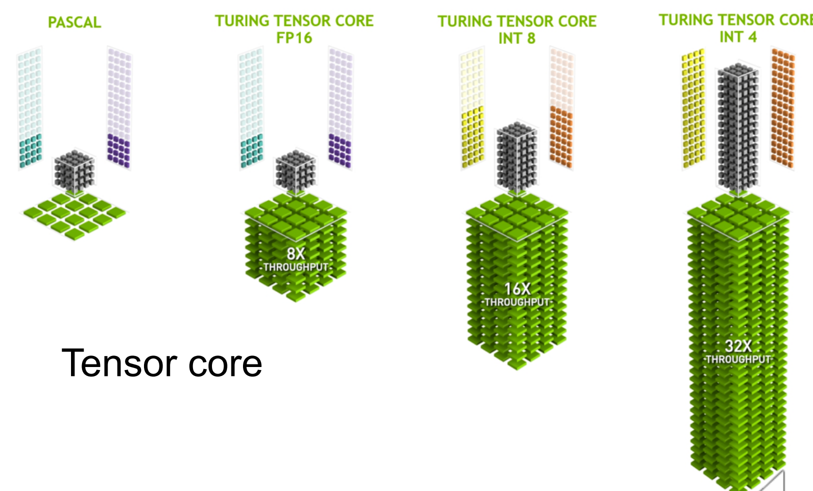


Advance in Graphics Processing Units (GPUs)



Over **60x** speedup in neural network training since 2013



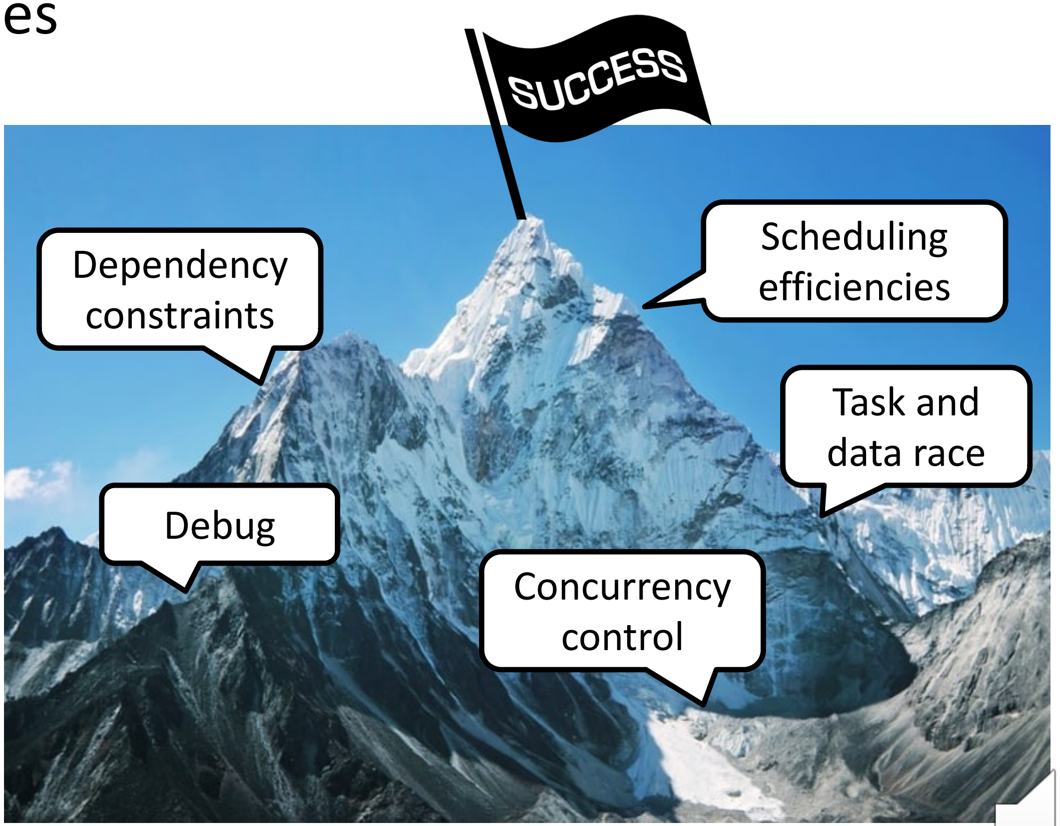


Tensor core

GPU Programming is NOT EASY

- **You need to deal with many difficult technical details**
  - Standard concurrency control
  - Task dependencies
  - Scheduling
  - Data layout
  - Kernel launch
  - … (more)

Many developers have hard time in getting them right!



Well, We have Seen Vast Success in ML

Machine learning (ML) frameworks judiciously hide implementation complexities of GPU parallelism!

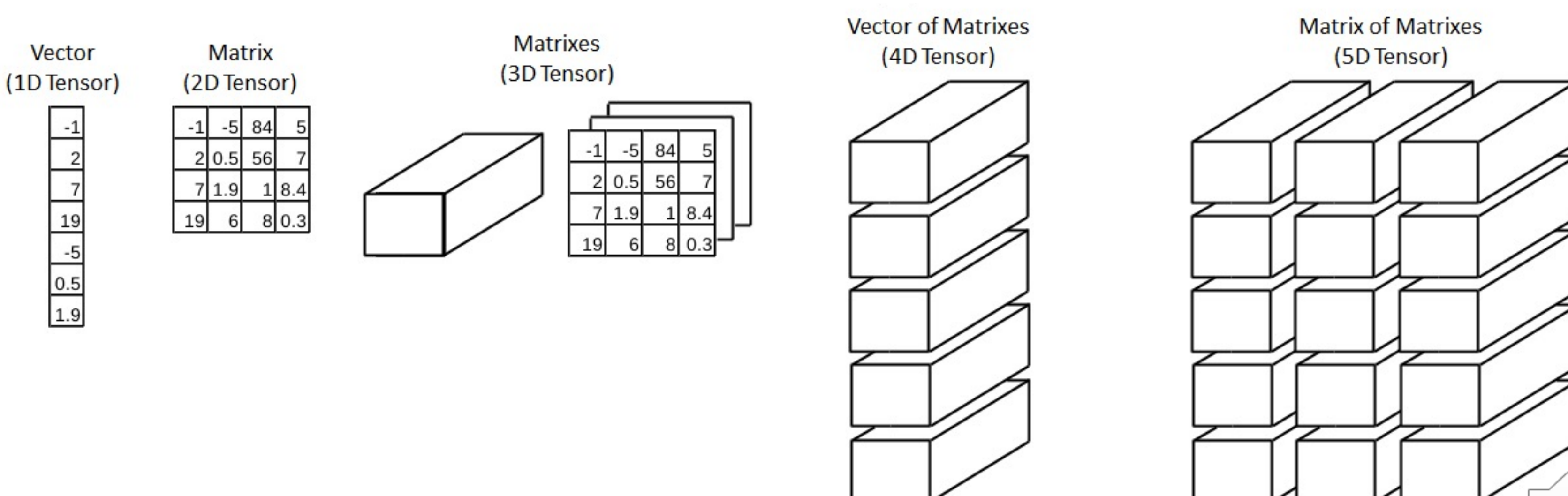
DEEP LEARNING

You don't need to know much GPU programing when running ML algorithms

How can we take advantage of existing deep learning frameworks to create GPU-accelerated EDA algorithms?

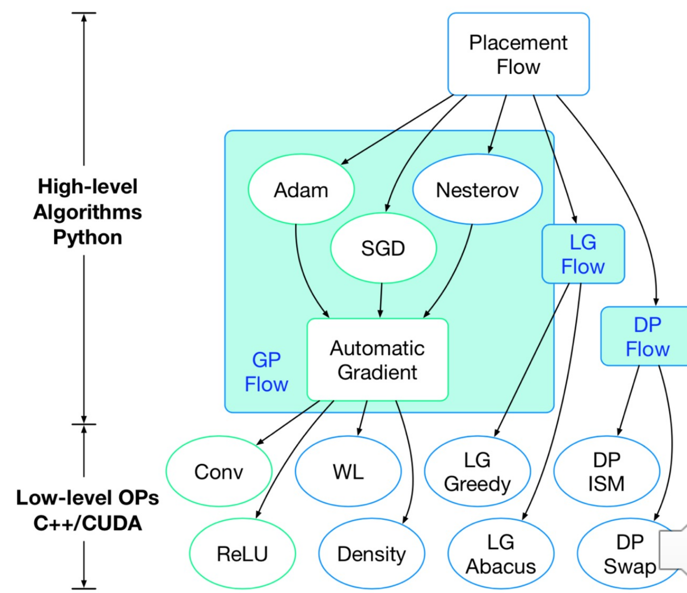
Key Abstraction of Deep Learning Frameworks

- **All deep learning frameworks rely on ”tensor”**
- **What is a tensor?**
  - A multidimensional view of a data layout
  - A unified data abstraction to utilize GPU
  - A GPU-efficient data representation



ML System-Enabled GPU Acceleration for EDA

- ❑ **Rethink an EDA algorithm from tensor's perspective**
  - ❑ Flat the data layout into N-dimensional arrays
  - ❑ Design algorithms on top these array
  - ❑ Reuse existing GPU facility in ML frameworks
- ❑ **We have seen some successful examples**
  - ❑ DREAMPlace (DAC19)
  - ❑ ABCDPlace (TCAD20)
  - ❑ RTL simulation (ICCAD20)
  - ❑ …

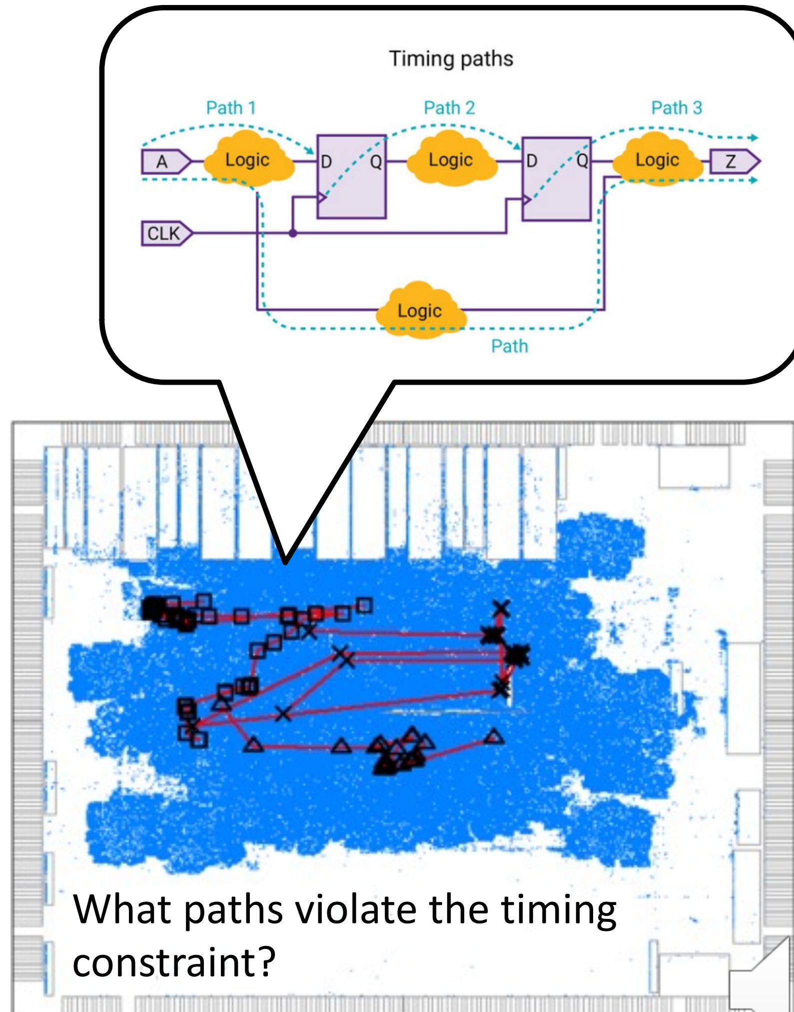


Case Study

1. Z Guo, T-W Huang, and Y Lin, "GPU-Accelerated Static Timing Analysis," *IEEE/ACM ICCAD*, 2020
2. G Guo, T-W Huang, Y Lin, and M Wong, "GPU-Accelerated Path-based Timing Analysis," *IEEE/ACM DAC*, 2021

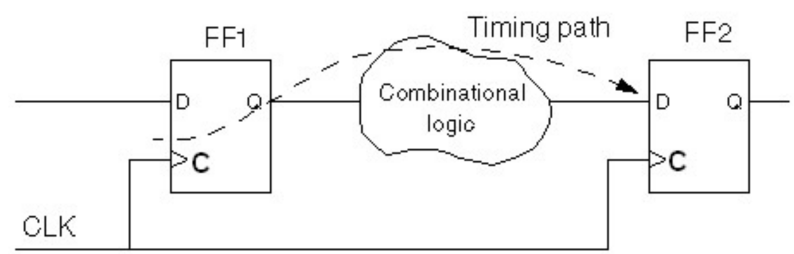
Static Timing Analysis

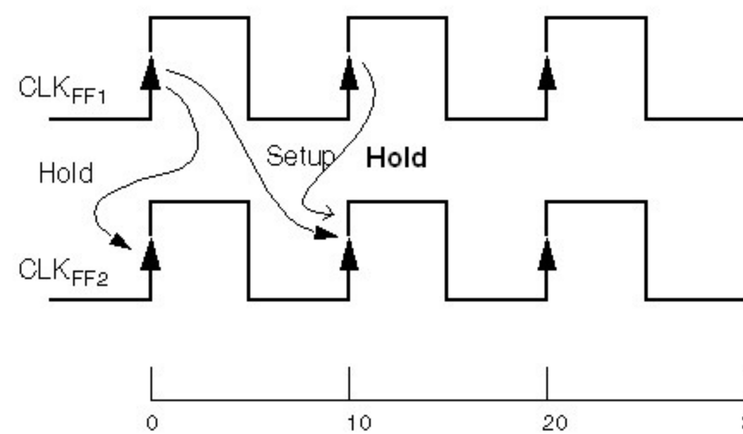
- **Static timing analysis (STA)**
  - Key step in the VLSI design
  - Verify the circuit timing
- **Analyze worst-case timing**
  - Minimum timing values
  - Maximum timing values
- **Challenges**
  - Compute giant graphs
  - Analyze millions of paths
  - Balance the loads
  - …

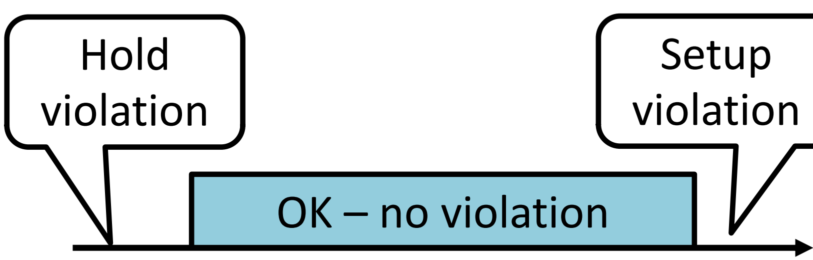


Timing Checks (Required Arrival Time)

- Modern circuits are sequential
  - Drive data signal via clocks
  - Capture data via flip-flops (FF)s
- Timing constraints
  - Min required arrival time
    - After clock: hold
  - Max required arrival time
    - Before clock: setup



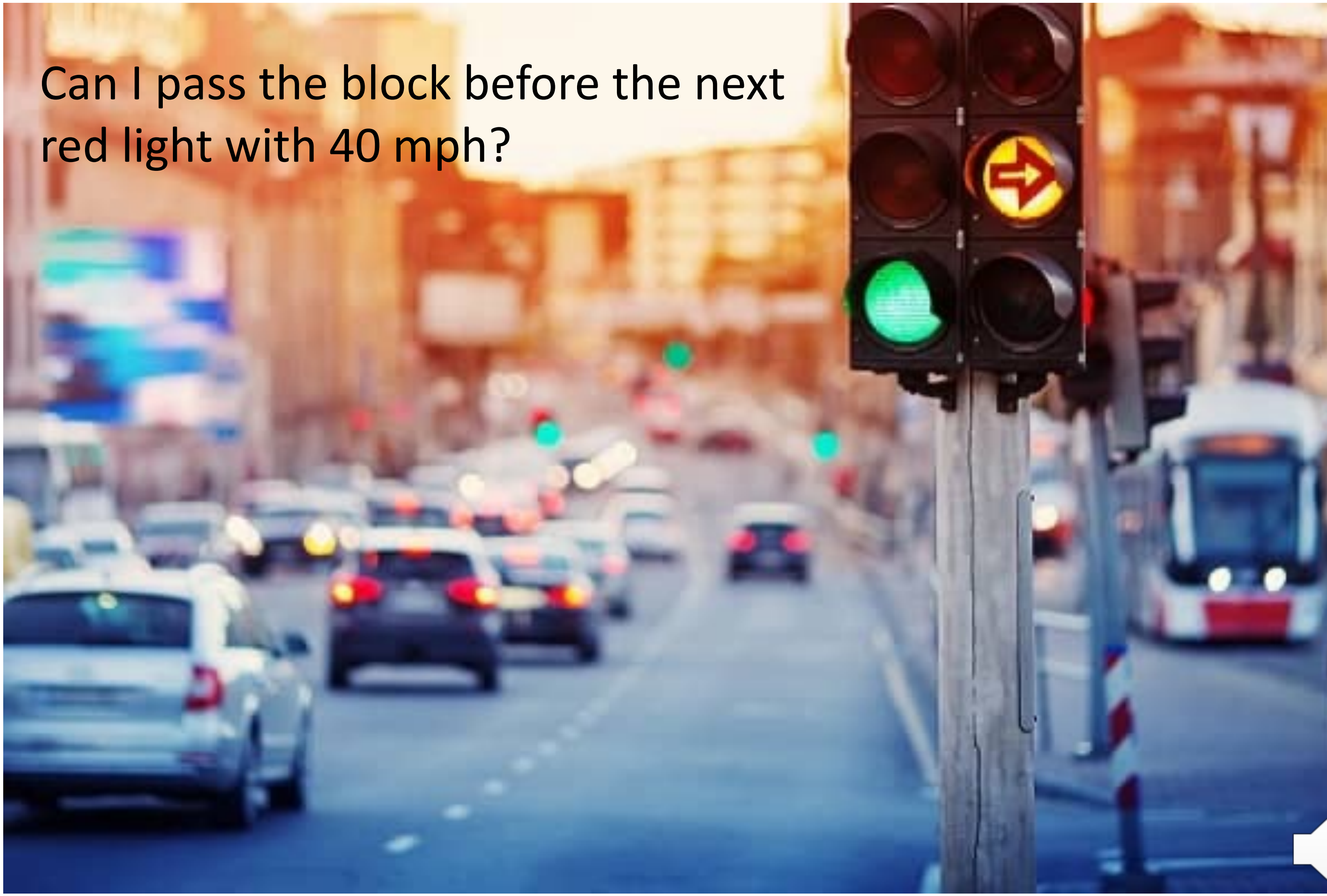




Required arrival time interval

The “Traffic Light” Analogy

Can I pass the block before the next red light with 40 mph?

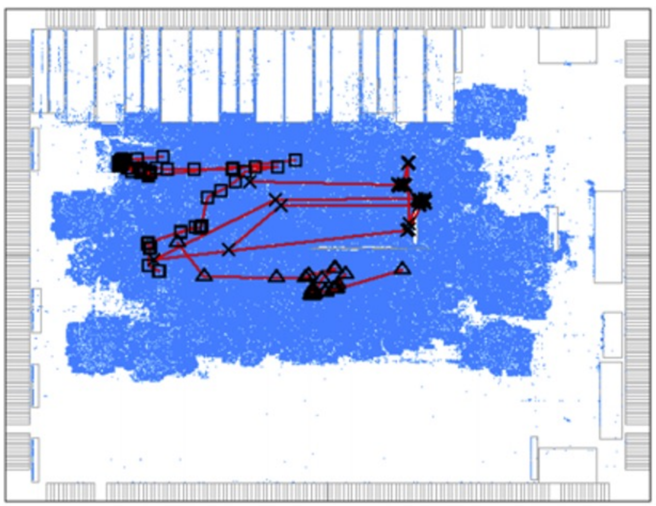


Building a Good Traffic System is Hard

- ❑ **Trillions of sections and traffic lights to analyze …**

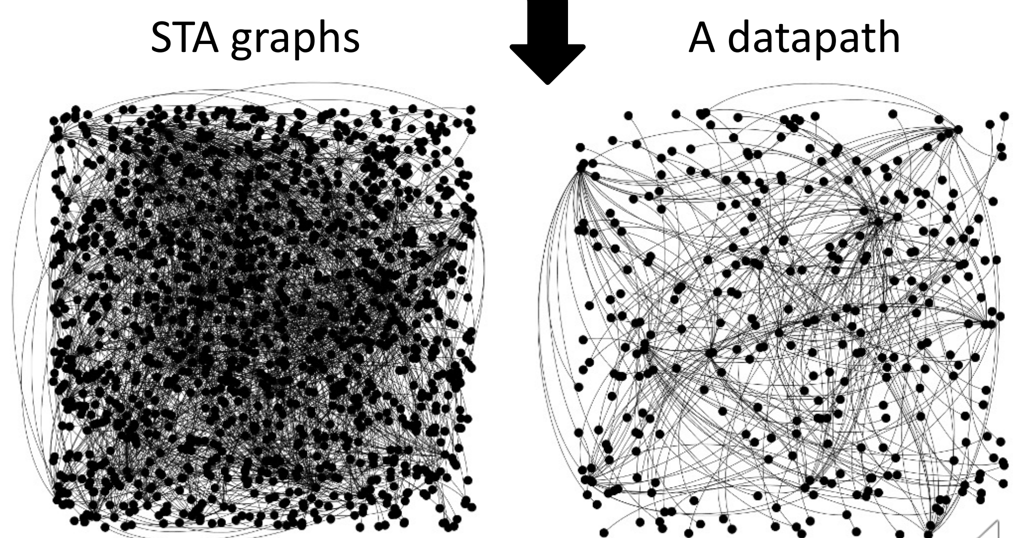
Same, STA is Computationally Challenging

- ❑ **STA graphs is extremely large and irregular**
  - ❑ Millions to billions of nodes and edges
  - ❑ Propagate timing information along giant graphs



ISPD circuit design (10M gates)

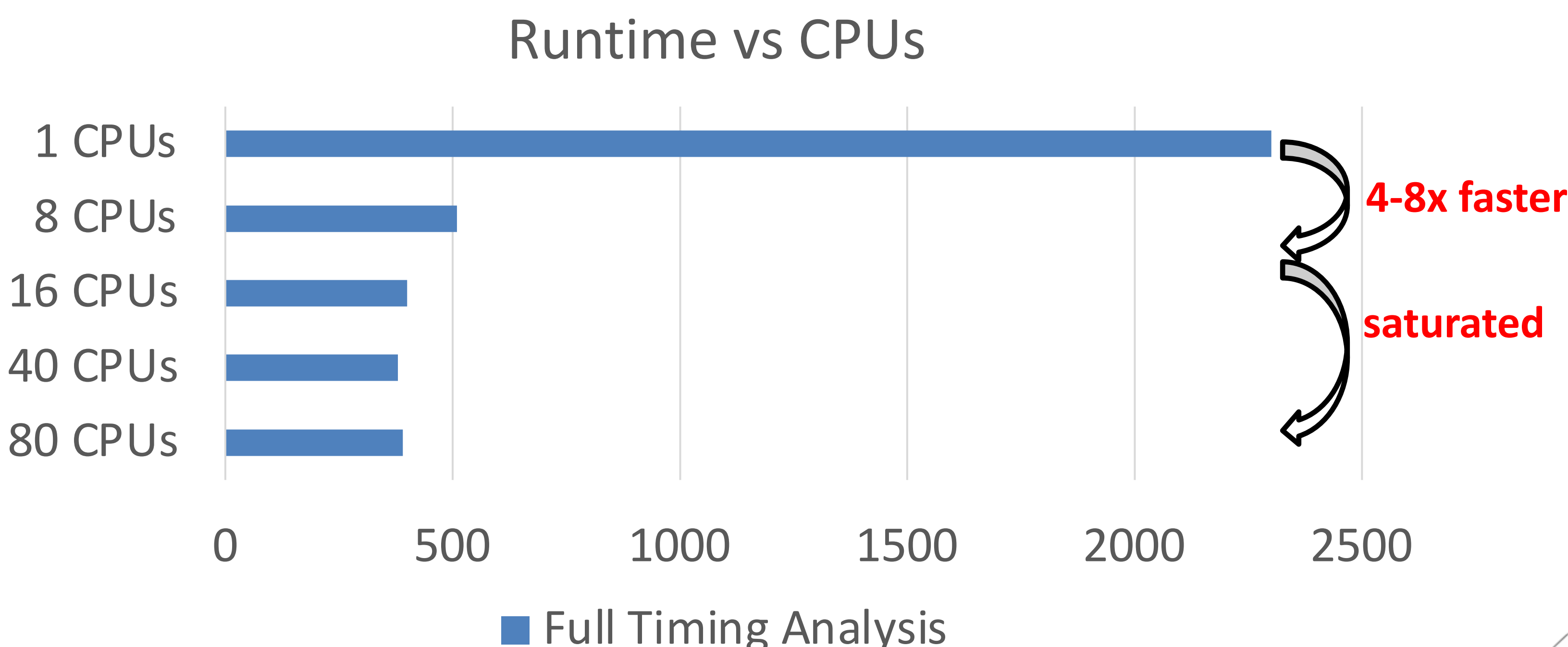
Complete analysis can take **8 hours** and **800 GB RAM**



STA graphs are extremely large and irregular

Parallel Timing Analysis is a MUST

- **Leverage many-core CPUs to speed up the runtime**
  - Dramatic speed-up using 8 cores
  - Yet, scalability saturates at about 10—16 cores



Observed Scalability Bottleneck

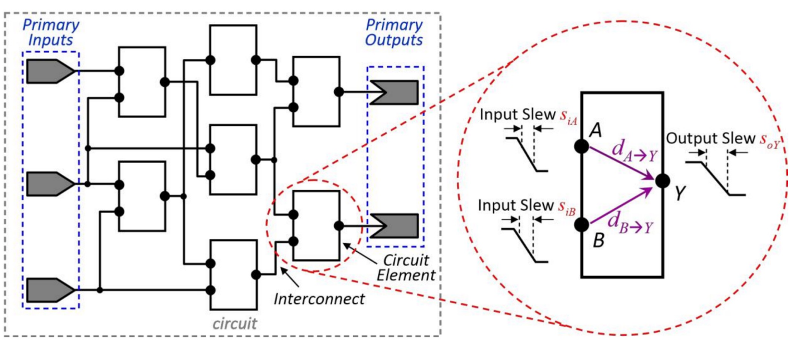
- **CPU-only parallelism stagnates at about 10 cores**
  - "Amdahl's Law" limits the strong scalability
  - Circuit graph structures limits the maximum parallelism
    - If the graph has only 10 parallel nodes at a level, we won't achieve 40x speed-up
  - Irregular computations limits the memory bandwidth
    - STA is graph-oriented, not cache-friendly
- **Need to incorporate new parallel paradigms**
  - GPU opens opportunities for new scalability milestones
    - e.g., 100x speed-up reported in logic simulation
    - e.g., 20—80x speed-up reported in placement
  - Implement our algorithms using PyTorch's tensor library

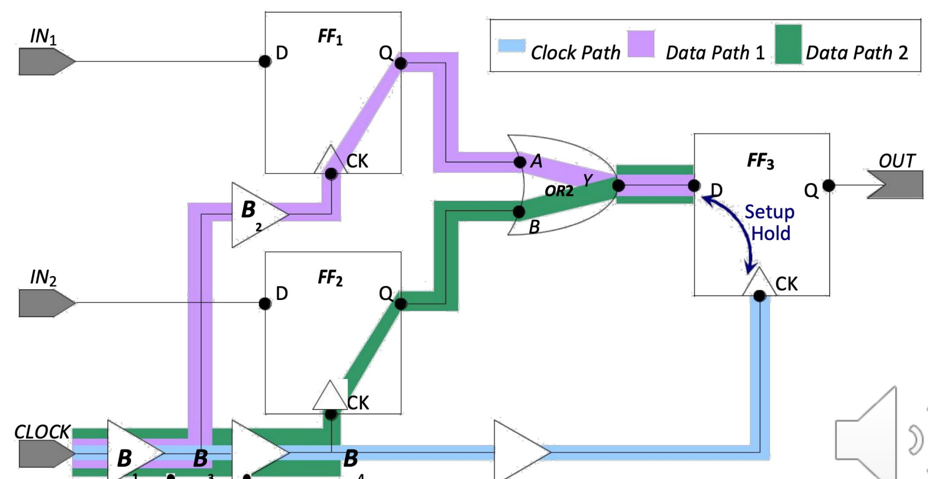
Leverage GPU to Accelerates STA

- **We target two important STA steps:**
  - Graph-based analysis (GBA)
  - Path-based analysis (PBA)
- **We design CPU-GPU collaborative STA algorithms**
  - CPU-GPU task decomposition
  - GPU kernels for timing update

PBA analyzes critical paths one by one on a updated graph

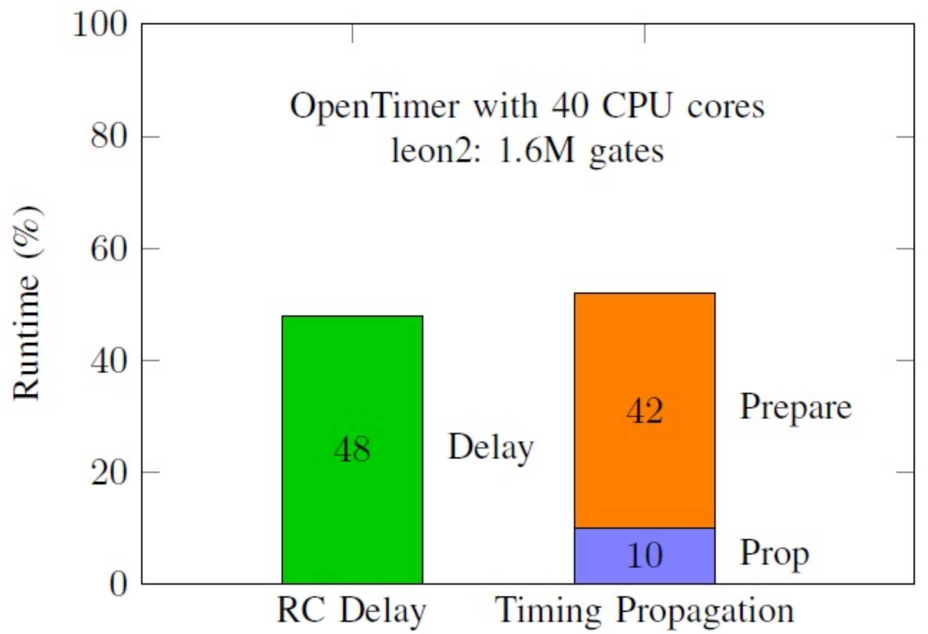
GBA computes the delay, slew, arrival time at each node and edge



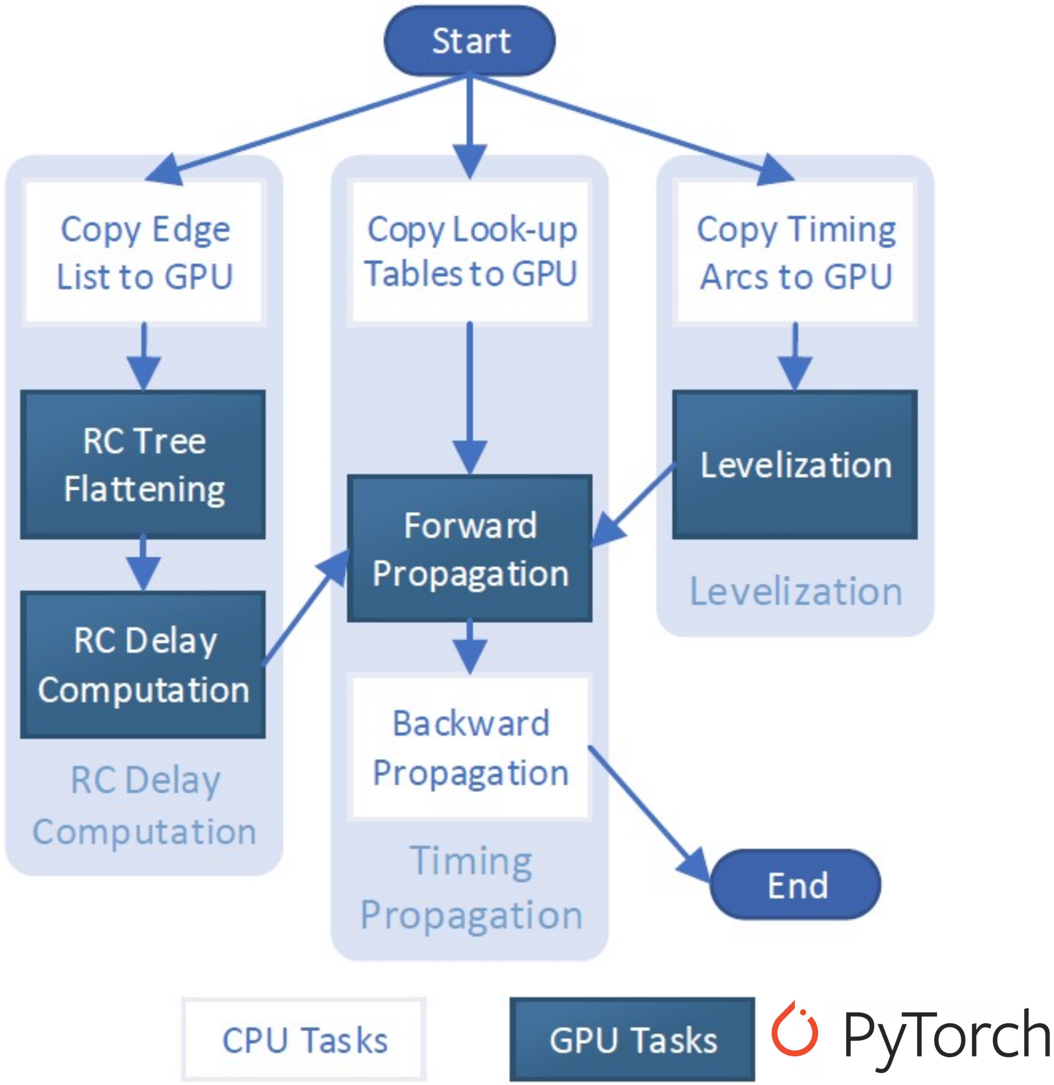


Runtime Breakdown of GBA

- **GBA has three time-consuming steps**
  1. Prepare tasks through levelization → 42% runtime
  2. Compute RC delay → 48% runtime
  3. Propagate timing → 10% runtime

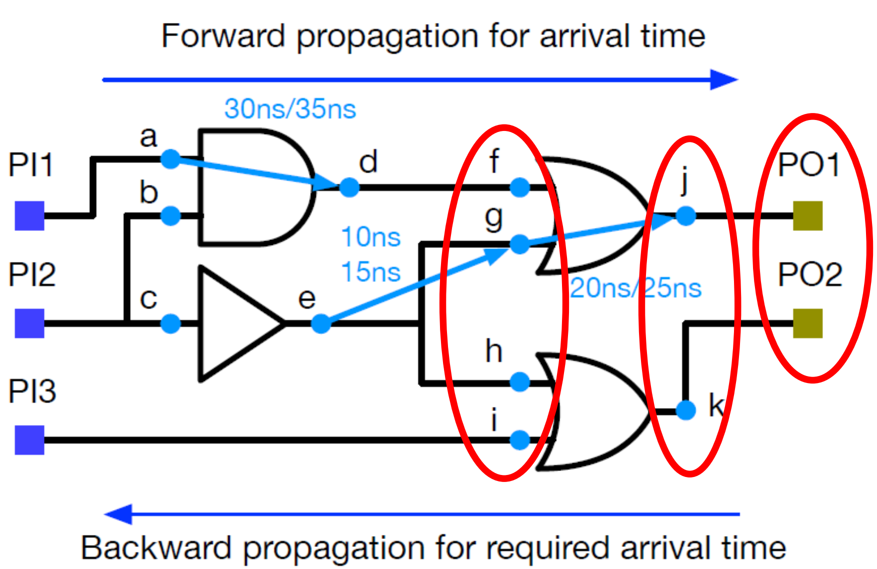


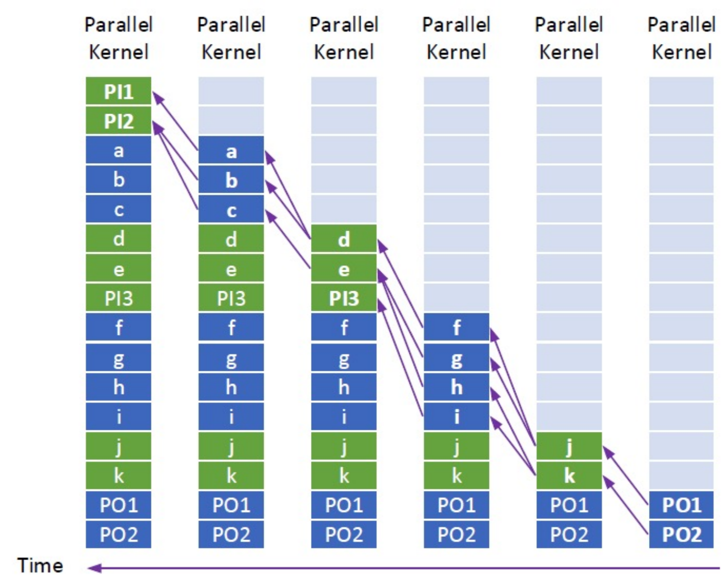
GPU-Accelerated GBA Algorithm Flow



Step #1: Levelization

- Levelize the circuit graph to a 2D levellist
  - Nodes at the same level can run in parallel (red circle)
  - Nodes at the same level can be modeled as a batch

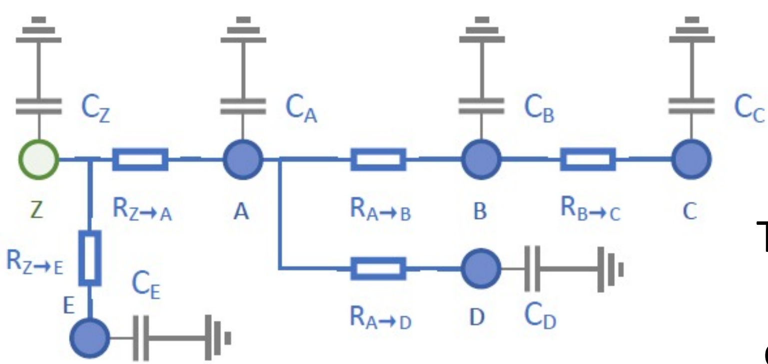




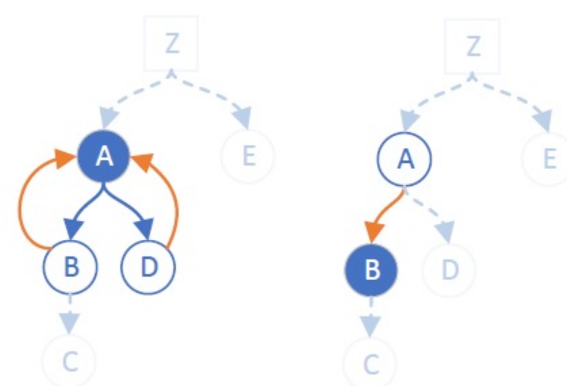
- GPU-accelerated levelization using parallel frontiers

Step #2: RC Update

- **The Elmore delay model**
- **Phase 1:** $load_u = \sum_{v \text{ is child of } u} cap_v$
  - For example, $load_A = cap_A + cap_B + cap_C + cap_D = cap_A + load_B + load_D$
- **Phase 2:** $delay_u = \sum_{v \text{ is any node}} cap_v \times R_{Z \to LCA(u,v)}$
  - For example, $delay_B = cap_A R_{Z\to A} + cap_D R_{Z\to A} + cap_B R_{Z\to B} + cap_C R_{Z\to B} = delay_A + R_{A\to B} load_B$



Two-phase tree traversal to compute delay



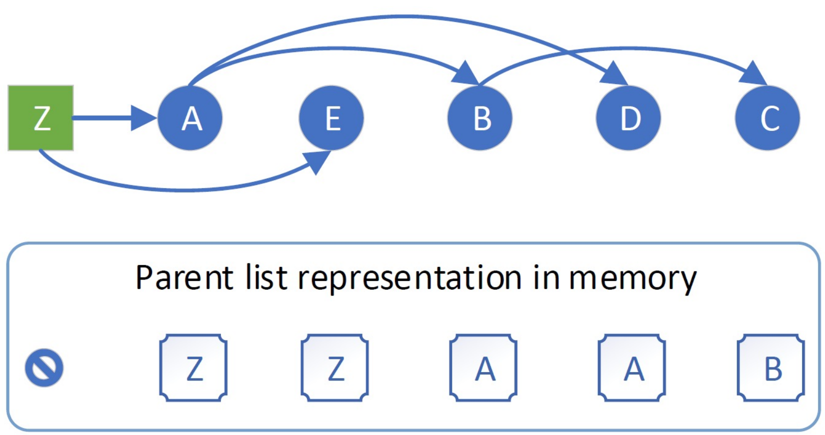
(a) Upward (b) Downwar

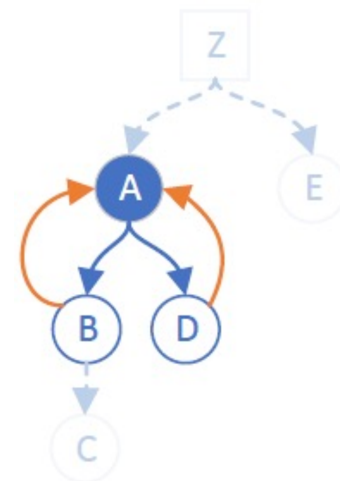
Step #2: RC Update Upward Phase

- ❑ **Store the parent index of each node on GPU**
- ❑ **Perform 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]
```





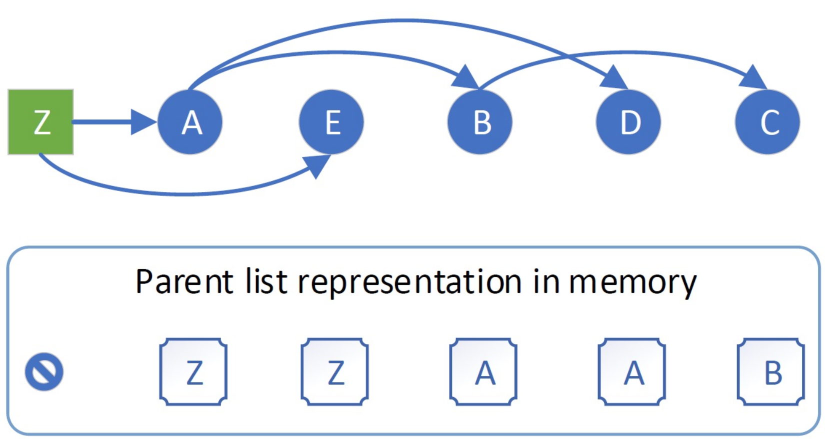
(a) Upward

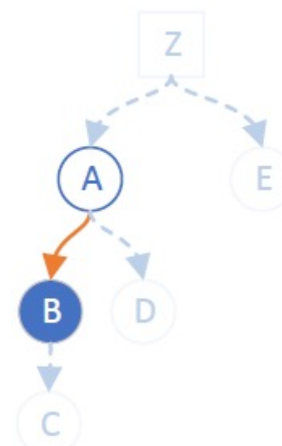
Step #2: RC Update Downward Phase

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

```
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
```

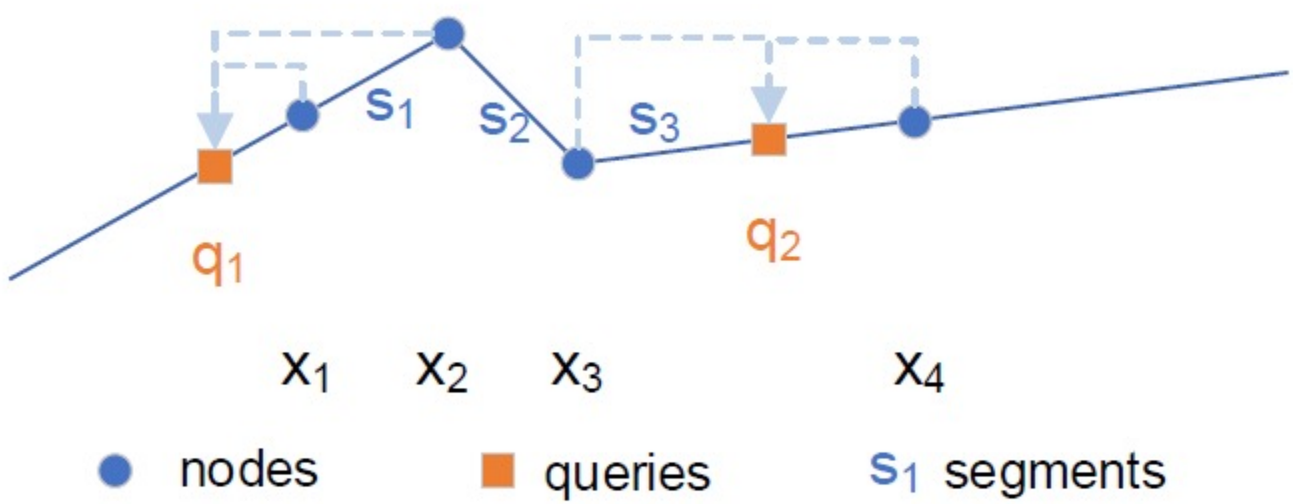




(b) Downward

Step #3: Cell Delay Update

- **Perform linear inter- and extra-polation in batches**
  - x-axis and then y-axis



Overall Performance

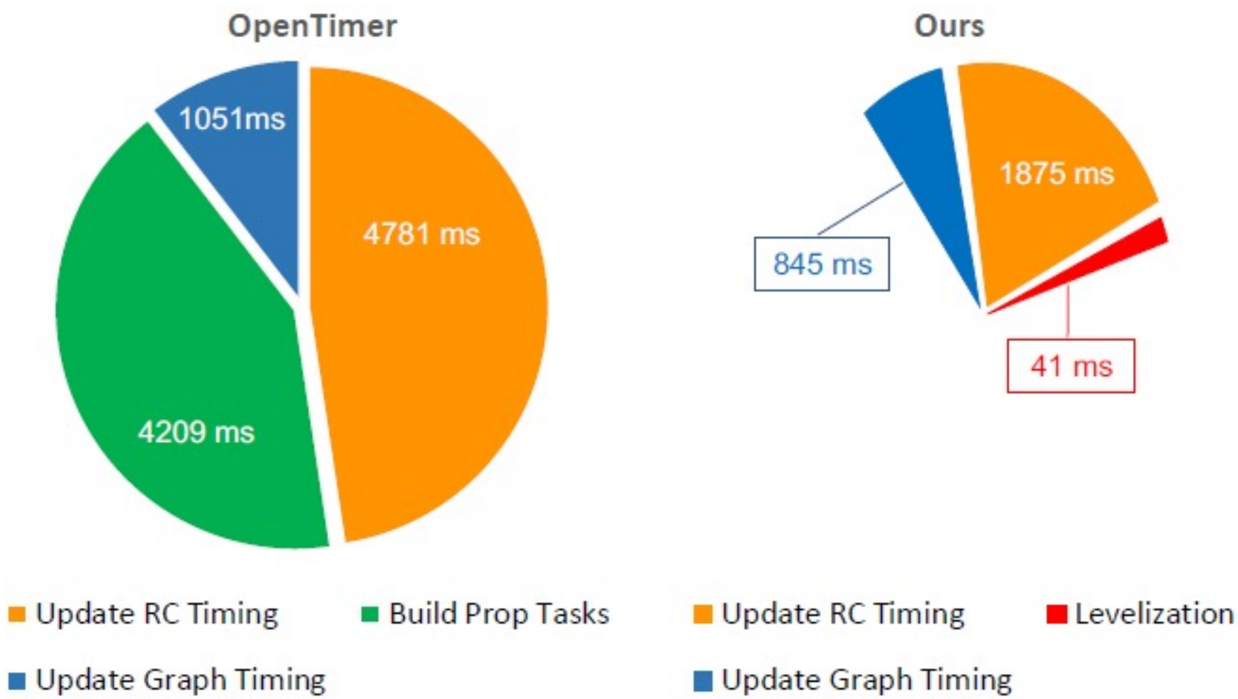
- **Implemented based on PyTorch's Tensor Library**
- **Comparison with OpenTimer of 40 CPUs**
  - Run on large TAU15 Benchmarks (>20K gates)
  - Run on one Nvidia RTX 2080

Benchmark	# PIs	# POs	# Gates	# Nets	# Pins	# Nodes	# Edges	OpenTimer Runtime (40 CPUs)	Our Runtime (40 CPUs 1 GPU)	
									Runtime	Speed-up
aes_core	260	129	22938	23199	66751	413588	453508	156 ms	138 ms	1.13×
vga_lcd	85	99	139529	139635	397809	1966411	2185601	829 ms	311 ms	2.67×
vga_lcd_iccad	85	99	259067	259152	679258	3556285	3860916	1480 ms	496 ms	2.98×
b19	22	25	255278	255300	782914	4423074	4961058	1831 ms	585 ms	3.13×
cordic	34	64	45359	45393	127993	7464477	820763	274 ms	167 ms	1.64×
des_perf	234	140	138878	139112	371587	2128130	2314576	832 ms	325 ms	2.56×
edit_dist	2562	12	147650	150212	416609	2638639	2870985	1059 ms	376 ms	2.86×
fft	1026	1984	38158	39184	116139	646992	718566	241 ms	148 ms	1.63×
leon2	615	85	1616369	1616984	4328255	22600317	24639340	10200 ms	2762 ms	3.69×
leon3mp	254	79	1247725	1247979	3376832	17755954	19408705	7810 ms	2585 ms	3.02×
netcard	1836	10	1496719	1498555	3999174	21121256	23027533	9225 ms	2571 ms	3.60×
mgc_edit_dist	2562	12	161692	164254	450354	2436927	2674934	1021 ms	368 ms	2.77×
mgc_matrix_mult	3202	1600	171282	174484	492568	2713241	2994343	1138 ms	377 ms	3.02×
tip_master	778	857	37715	38493	95524	533690	570154	163 ms	143 ms	1.14×

 # PIs: number of primary inputs **# POs**: number of primary outputs **# Gates**: number of gates **# Nets**: number of nets
 # Pins: number of pins **# Nodes**: number of nodes in the STA graph **# Edges**: number of edges in the STA graph

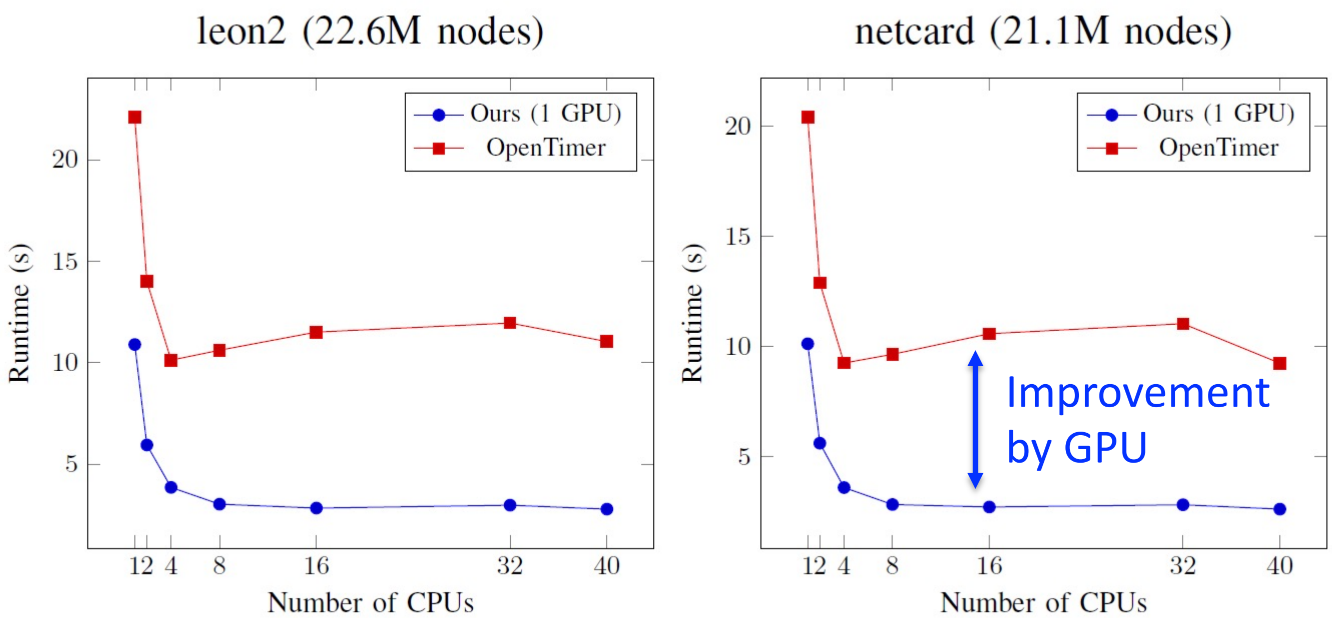
Runtime Breakdown

- Circuit leon2 (21 M nodes)



Runtime vs CPUs

- ❑ **Significant performance gap between CPU and GPU**

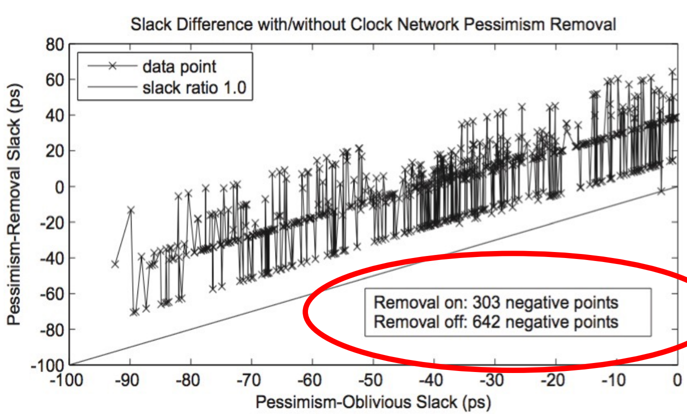


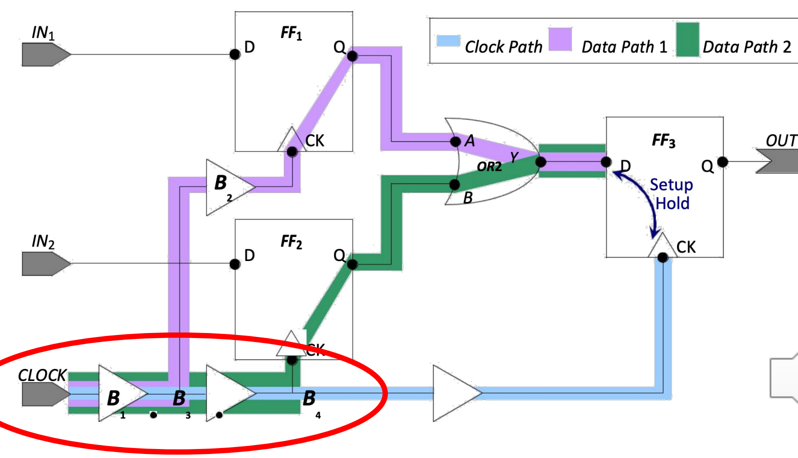
Our runtime of 1 CPU and 1 GPU is very close to OpenTimer of 40 CPUs

Path-based Analysis (PBA)

- **Identify a set of critical paths from a updated graph**
  - Exponential number of paths in the circuit graph
- **Re-analyze each path with path-specific update**
  - Re-propagate the slew and remove pessimism
  - Advanced on-chip variation (AOCV)
  - Common path pessimism removal (CPPR)
  - …

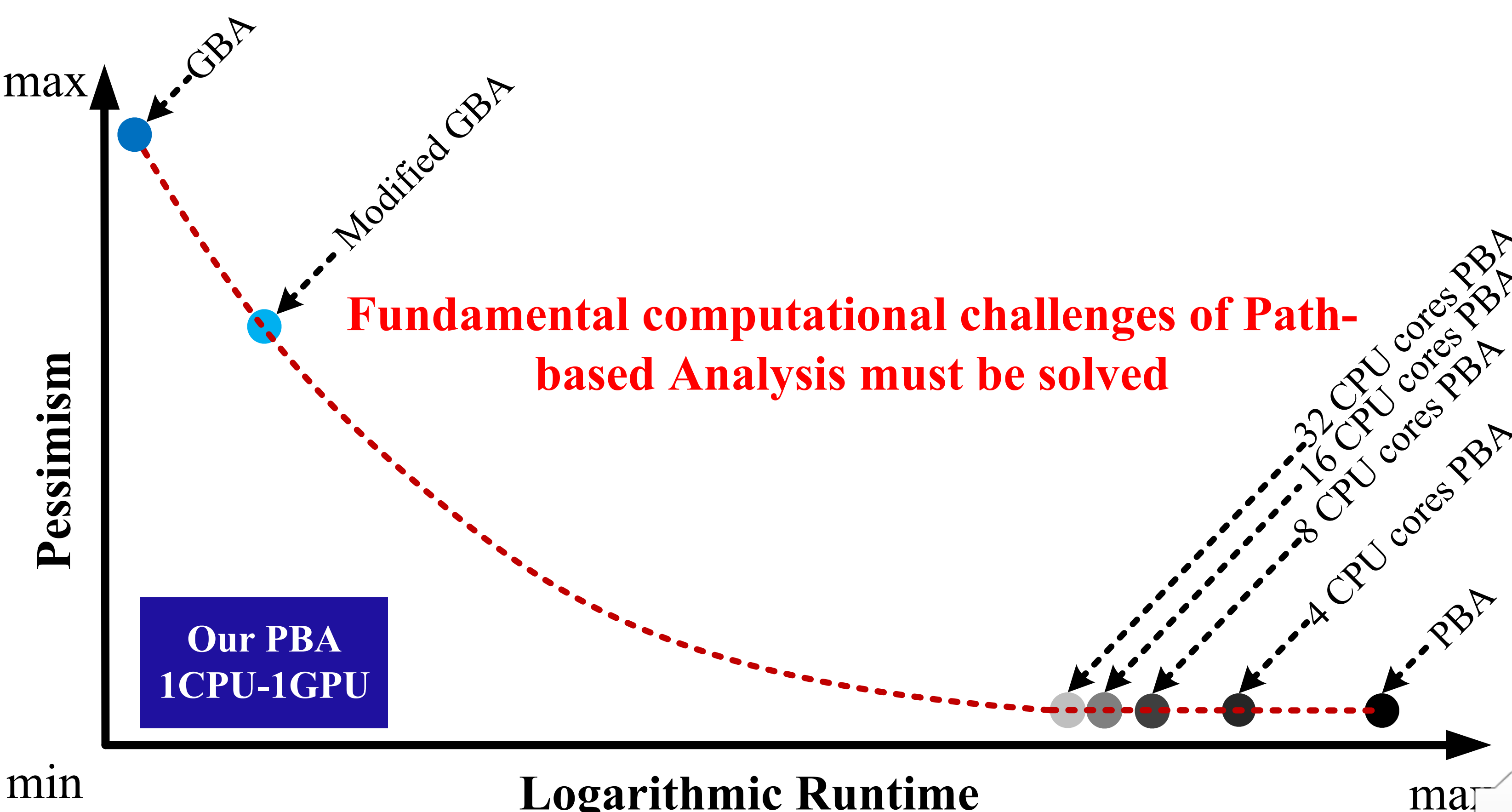
Paths marked failing at GBA may become passing after PBA!





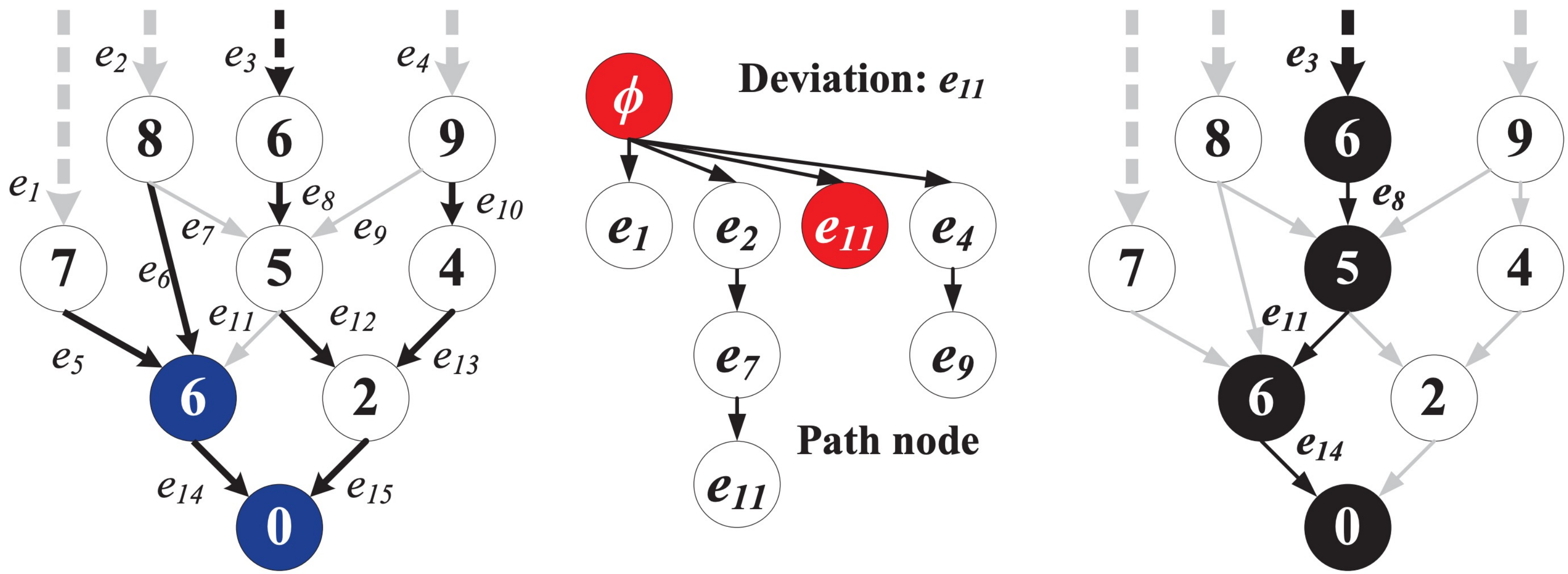
PBA is Extremely Time-Consuming

- Speed vs Accuracy (pessimism removal) tradeoff



A Key Step: Generate Critical Paths

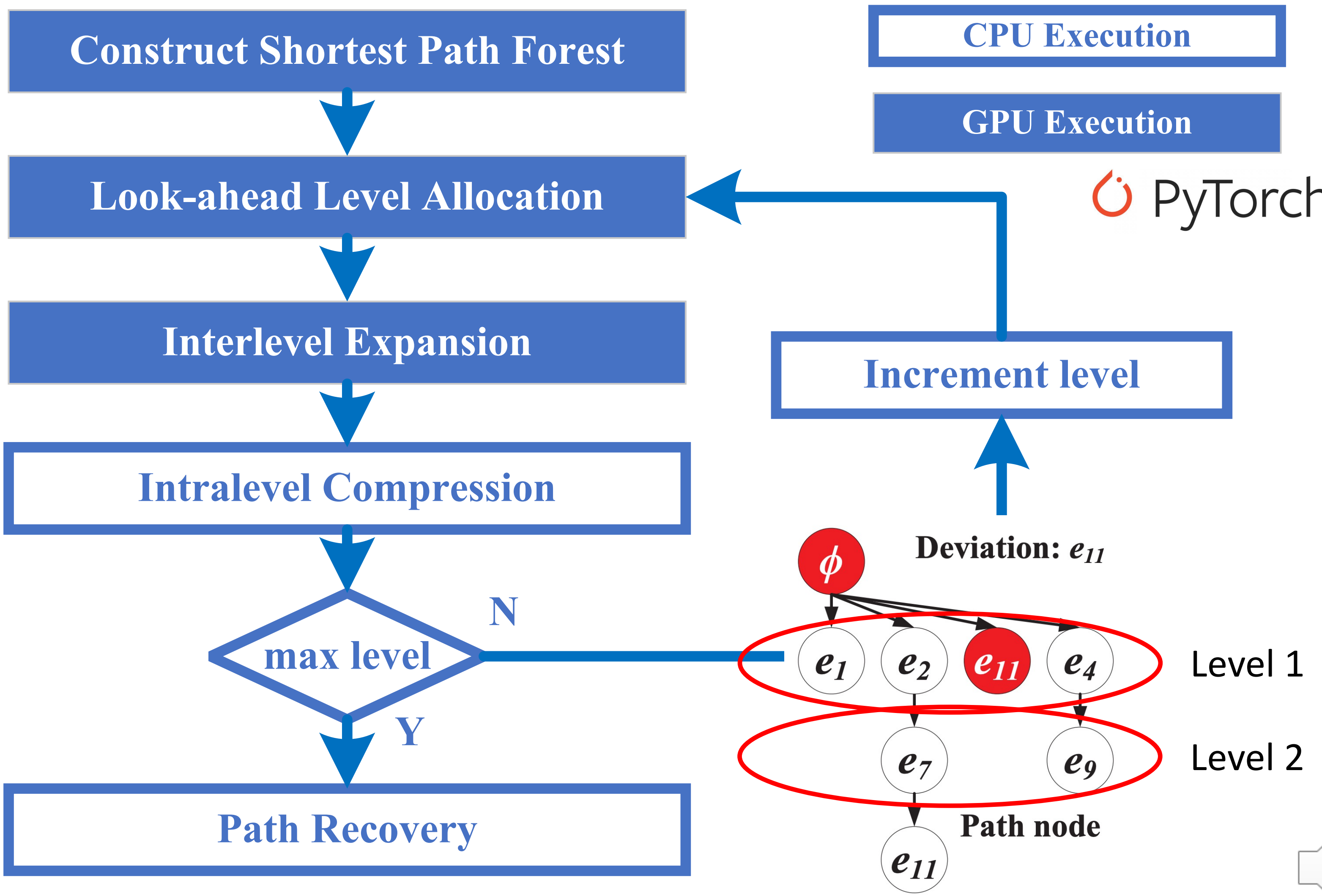
- ❑ **OpenTimer adopts implicit path representation**
  - ❑ Each path is represented using *O(1)* space and time
  - ❑ Each path is ranked through a *prefix* tree & a *suffix* tree



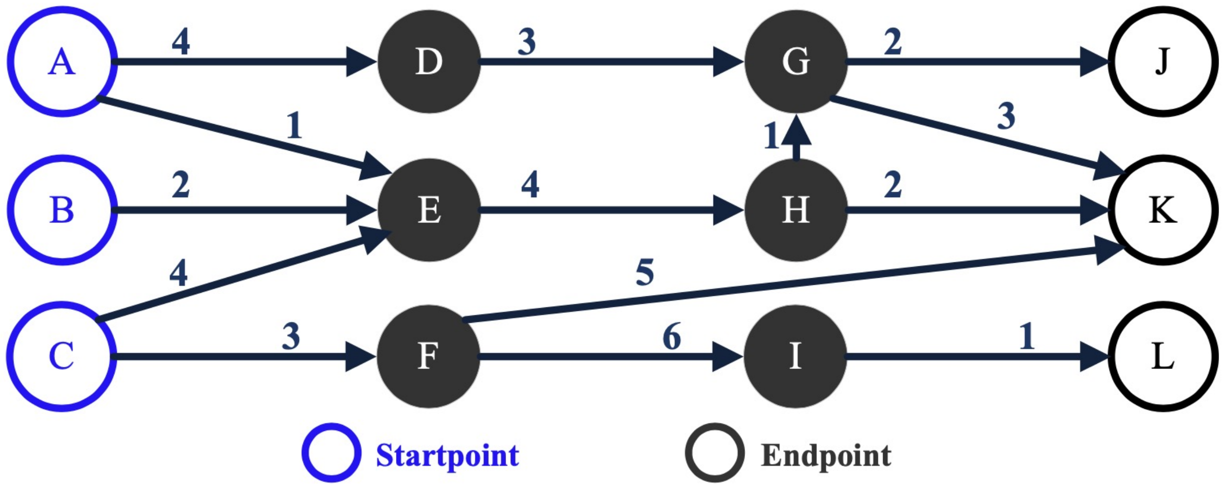
Path suffix: <e_{14}> + **Path prefix: <e_3, e_8, e_{11}>** = **Path: <e_3, e_8, e_{11}, e_{14}>**

T.-W. Huang et al., "OpenTimer v2: A New Parallel Incremental Timing Analysis Engine," IEEE TCAD '21

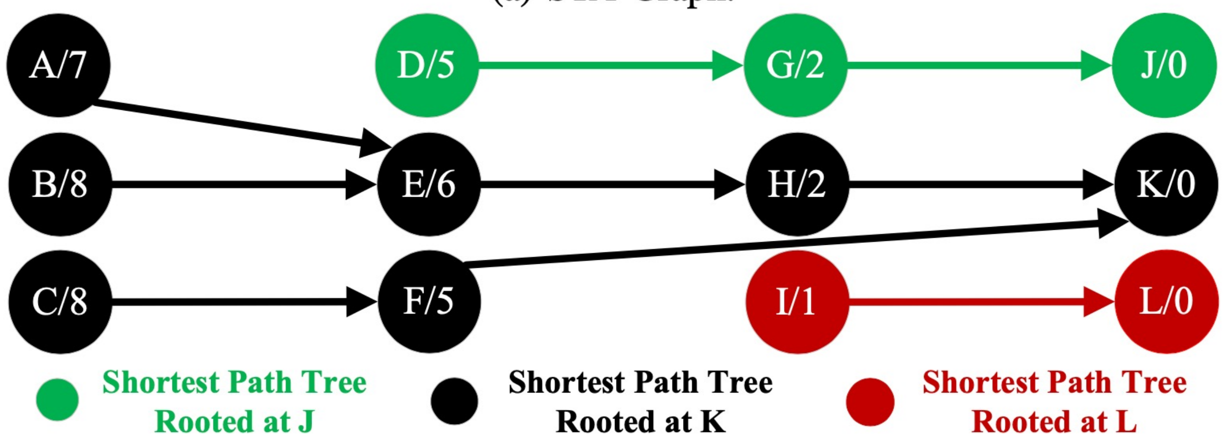
GPU-Accelerated PBA Algorithm Flow



Step #1: Generate Suffix Tree on GPU

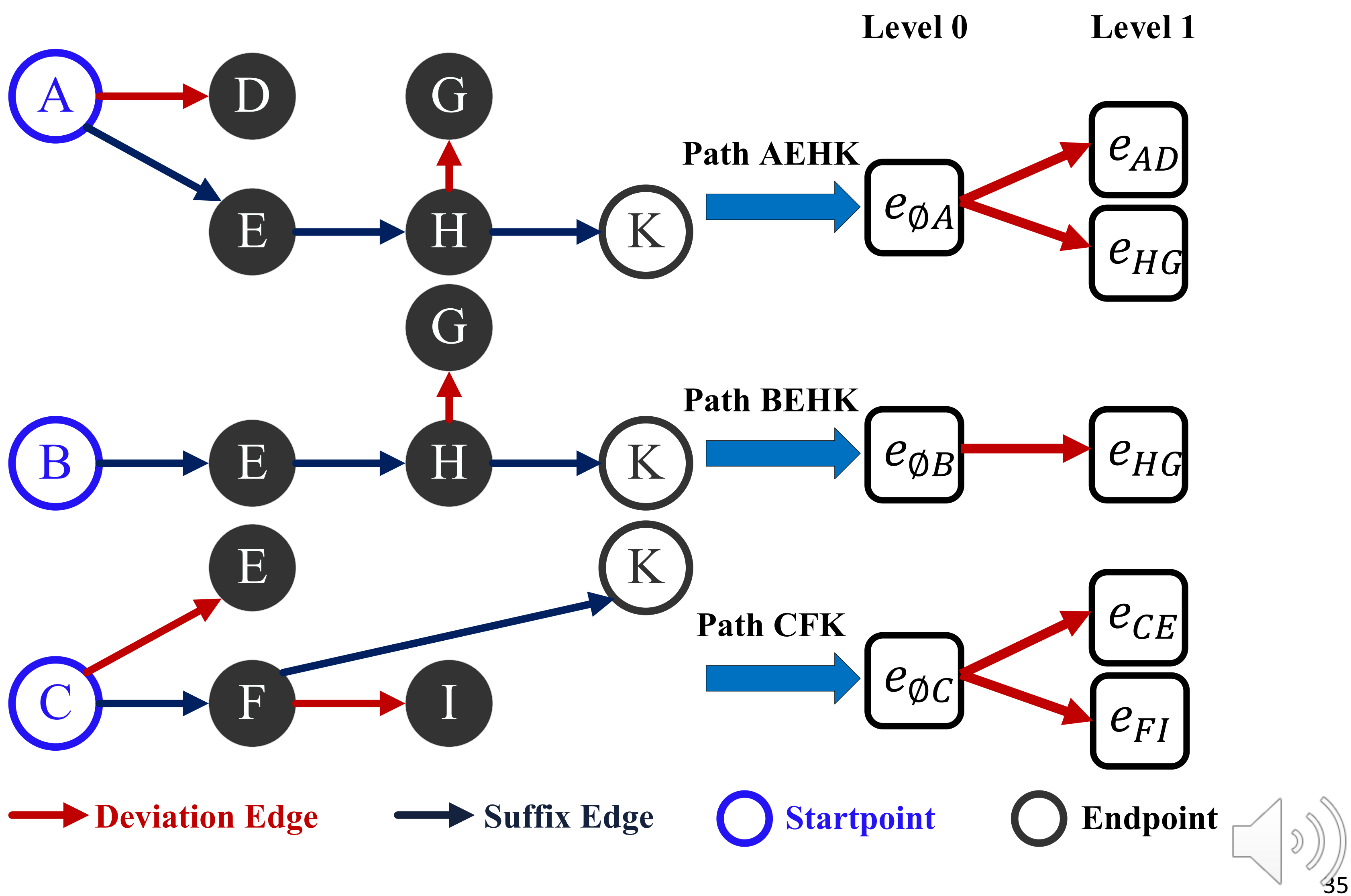


(a) STA Graph.



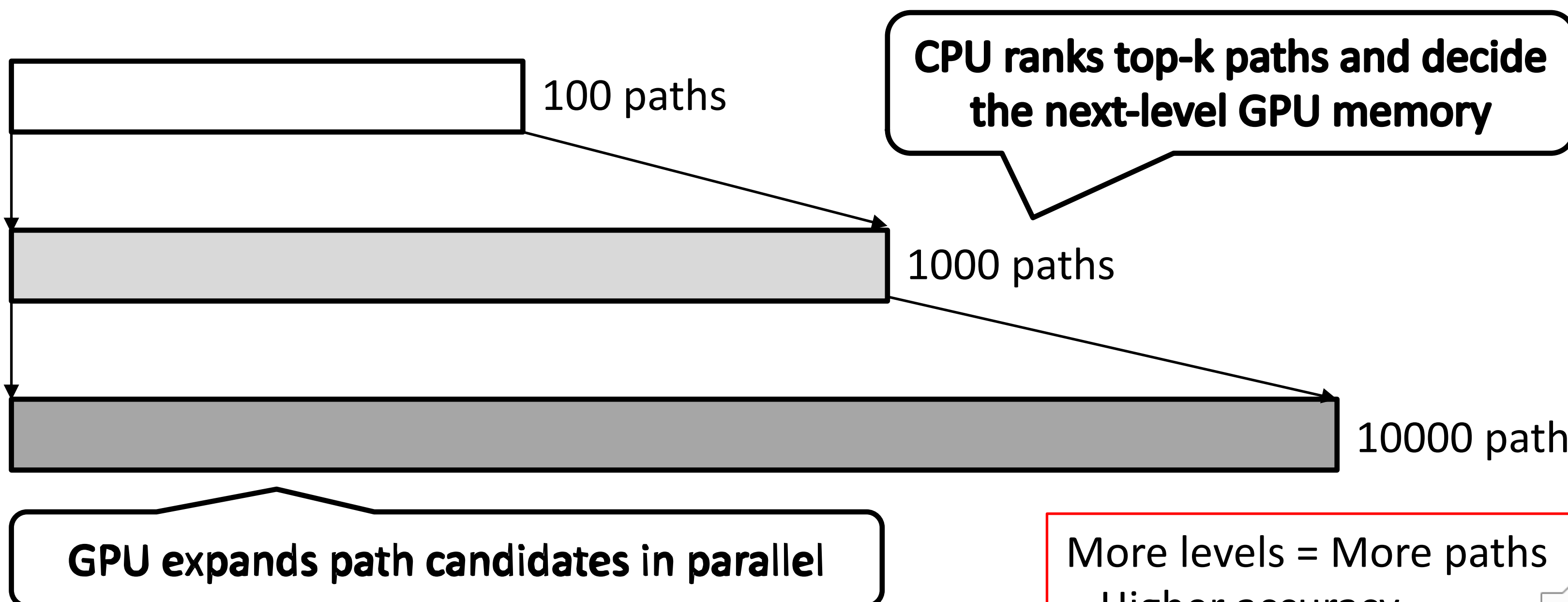
(b) Shortest path forest.

Step #2: Expand Prefix Tree on GPU



Step #2: Expand Prefix Tree on GPU (cont'd)

- **Iteratively grow GPU memory at each expansion**
  - Each iteration uses GPU to decide path candidates
  - Each iteration uses CPU to prune path candidates
  - Each path candidate takes O(1) space "deviation edge"



More levels = More paths
= Higher accuracy

Overall Performance

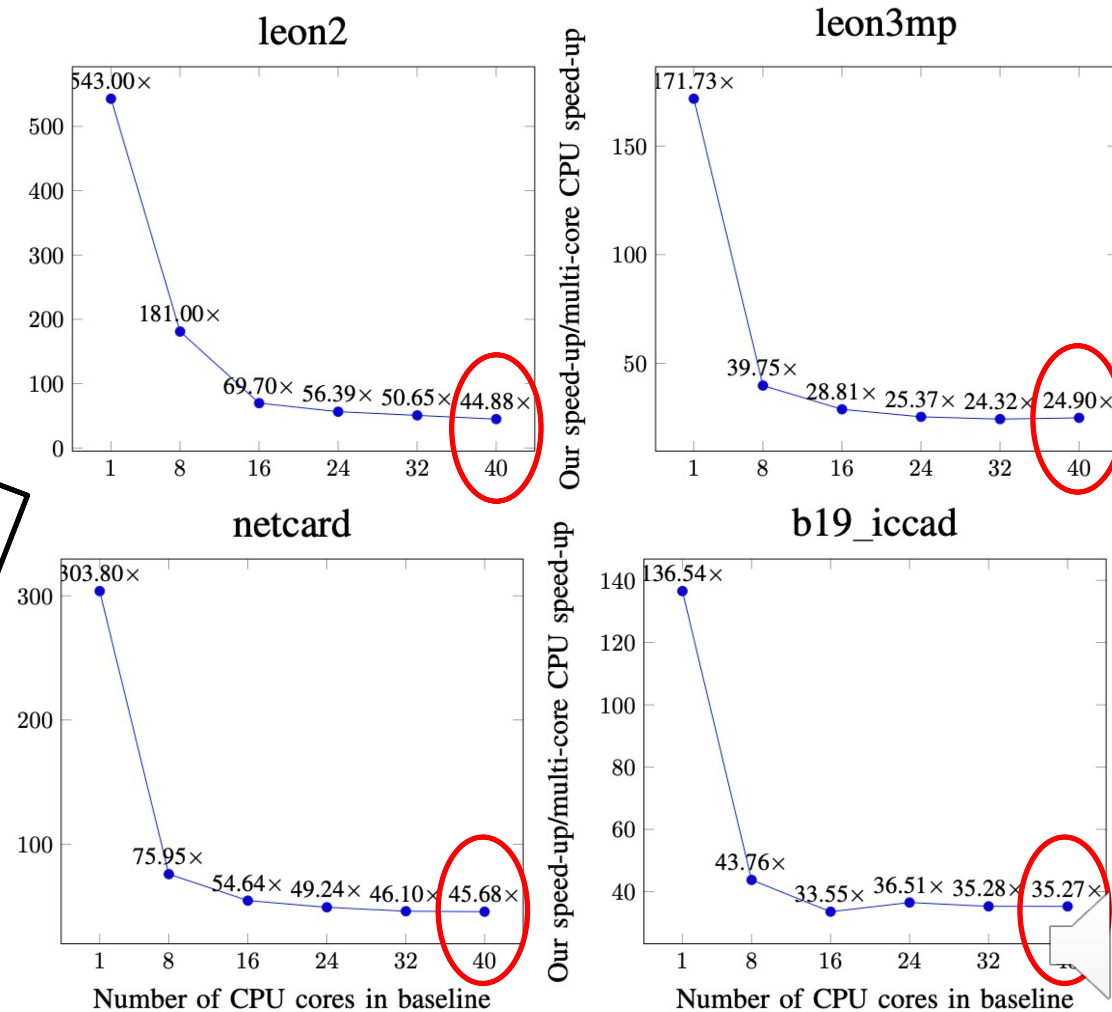
- **Implemented based on PyTorch's Tensor library**
- **Compare with OpenTimer's CPU-based PBA**
  - Report speed-up at different MDLs

Benchmark	#Pins	#Gates	#Arcs	OpenTimer Runtime	Our Algorithm #MDL=10		Our Algorithm #MDL=15		Our Algorithm #MDL=20	
					Runtime	Speed-up	Runtime	Speed-up	Runtime	Speed-up
leon2	4328255	1616399	7984262	2875783	4708.36	611×	5295.49ms	543×	5413.84	531×
leon3mp	3376821	1247725	6277562	1217886	5520.85	221×	7091.79ms	172×	8182.84	149×
netcard	3999174	1496719	7404006	752188	2050.60	367×	2475.90ms	304×	2484.08	303×
vga_lcd	397809	139529	756631	53204	682.94	77.9×	683.04ms	77.9×	706.16	75.3×
vga_lcd_iccad	679258	259067	1243041	66582	720.40	92.4×	754.35ms	88.3×	766.29	86.9×
b19_iccad	782914	255278	1576198	402645	2144.67	188×	2948.94ms	137×	3483.05	116×
des_perf_ispd	371587	138878	697145	24120	763.79	31.6×	766.31ms	31.5×	780.56	30.9×
edit_dist_ispd	416609	147650	799167	614043	1818.49	338×	2475.12ms	248×	2900.14	212×
mgc_edit_dist	450354	161692	852615	694014	1463.61	474×	1485.65ms	467×	1493.90	465×
mgc_matric_mult	492568	171282	948154	214980	994.67	216×	1075.90ms	200×	1113.26	193×

- **Achieve significant speed-up at large designs**
  - 611x speed-up in leon2 (1.3M gates)
  - 221x speed-up in leon3mp (1.2M gates)

Path Accuracy vs MDL

- one GPU is even faster than OpenTimer with 40 CPUs
  - 44x on leon2
  - 25x on leon3mp
  - 46x on netcard
  - 35x on b19



In fact, according to our experiments, our GPU-accelerated PBA is always faster than OpenTimer's CPU baseline regardless of the core count

Conclusion

- **Introduced the runtime challenges of EDA**
  - EDA tools must incorporate new parallel paradigms to allow more efficient design space exploration and optimization
  - Deep learning systems can simplify the implementation complexities of GPU programming
- **Studied GPU-accelerated STA opportunities**
  - Graph-based analysis
  - Path-based analysis
- **Accelerated the graph-based analysis using GPU**
  - Achieved 4x speed-up on large designs
- **Accelerated the path-based analysis using GPU**
  - Achieved 600x speed-up on large designs