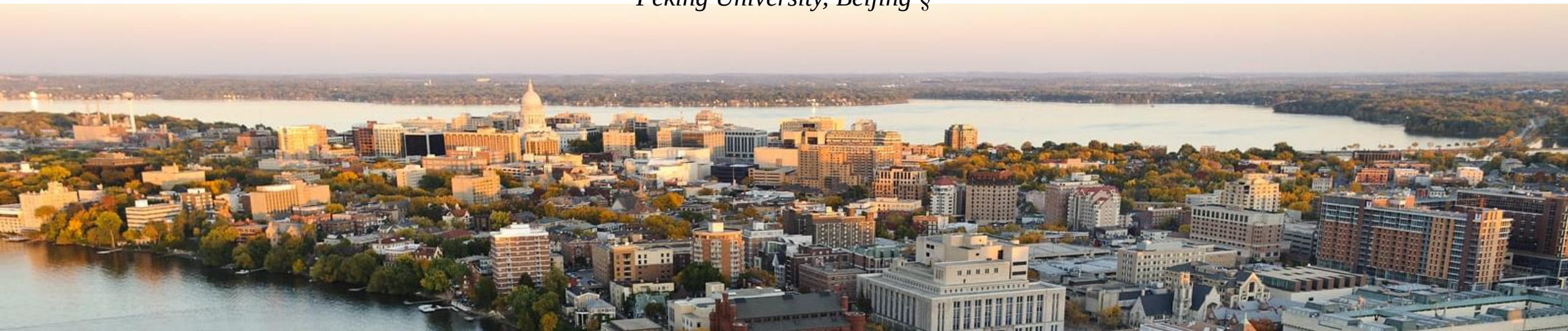
PathGen: An Efficient Parallel Critical Path Generation Algorithm

ASPDAC'25

Che Chang†, Boyang Zhang†, Cheng-Hsiang Chiu†,
Dian-Lun Lin†, Yi-Hua Chung†, Wan-Luan Lee†,
Zhizheng Guo§, Yibo Lin§, and Tsung-Wei Huang†

University of Wisconsin, Madison †

Peking University, Beijing §



What is Critical Path Generation? Why?

- What is Critical Path Generation (CPG)?
  - Given a directed-acyclic circuit graph, report the top-*k* critical paths in ascending order of path slack/delay
- Why is CPG important?
  - Crucial for **optimizing** and **verifying** circuit timing
  - Increasing design complexity makes CPG runtime a bottleneck in STA engines

Sequential CPG Algorithms

- **iTimerC**[1], **iitRace**[2], and **OpenTimer**[3] demonstrated good performance
- However, large CPG queries can be slow, impacting the performance of STA applications
    - e.g., a CPG query of 1M paths takes 2.5 seconds, where STA applications typically issue thousands of CPG queries

[1]P. -Y. Lee, I. H. -R. Jiang, C. -R. Li, W. -L. Chiu and Y. -M. Yang, "iTimerC 2.0: Fast incremental timing and CPPR analysis," *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Austin, TX, USA, 2015, pp. 890-894, doi: 10.1109/ICCAD.2015.7372665.

[2]C. Peddawad, A. Goel, Dheeraj B and N. Chandrachoodan, "iitRACE: A memory efficient engine for fast incremental timing analysis and clock pessimism removal," *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Austin, TX, USA, 2015, pp. 903-909, doi: 10.1109/ICCAD.2015.7372667.

[3]T. -W. Huang, G. Guo, C. -X. Lin and M. D. F. Wong, "OpenTimer v2: A New Parallel Incremental Timing Analysis Engine," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 4, pp. 776-789, April 2021, doi: 10.1109/TCAD.2020.3007319.

Multi-threaded CPG Algorithms

- Existing GPU-parallel CPG algorithm (Guo et al.[4])
  - Substantial runtime speedup (> 50×)
  - GPU support requires **significant investment** and **codebase modifications**
  - Some STA applications (e.g., incremental timing) **lack sufficient data parallelism** for GPU benefits
- A CPU-parallel CPG algorithm is needed to **co-enhance** the performance of STA applications

[4]G. Guo, T. -W. Huang, Y. Lin, Z. Guo, S. Yellapragada and M. D. F. Wong, "A GPU-Accelerated Framework for Path-Based Timing Analysis," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 11, pp. 4219-4232, Nov. 2023, doi: 10.1109/TCAD.2023.3272274.

CPU-parallel CPG is Challenging

- Cannot use the GPU approach out of the box
  - The parallelism model is totally different
  - GPU-specific data structure cannot be used
- Need to strategically **partition generated paths** into multiple groups to run in parallel
  - Also accommodate slack priorities
- Need to dynamically **re-balance slack priorities** in each partition
  - As more paths are generated, slack priorities becomes unbalanced in certain partitions, resulting in high contention

Technical Contributions

- We propose multi-level concurrent queue scheduling
  - To categorize generated critical paths into multiple queues
  - Paths in the same queue have **similar slacks** and can be **processed in parallel**
- We propose the geometric slack partitioning strategy
  - To **balance the path counts** in each queue and minimize contention
- We propose the node redistribution strategy
  - To **reassign slack priorities** to paths for more accurate results

Background

- Implicit path representation[3]

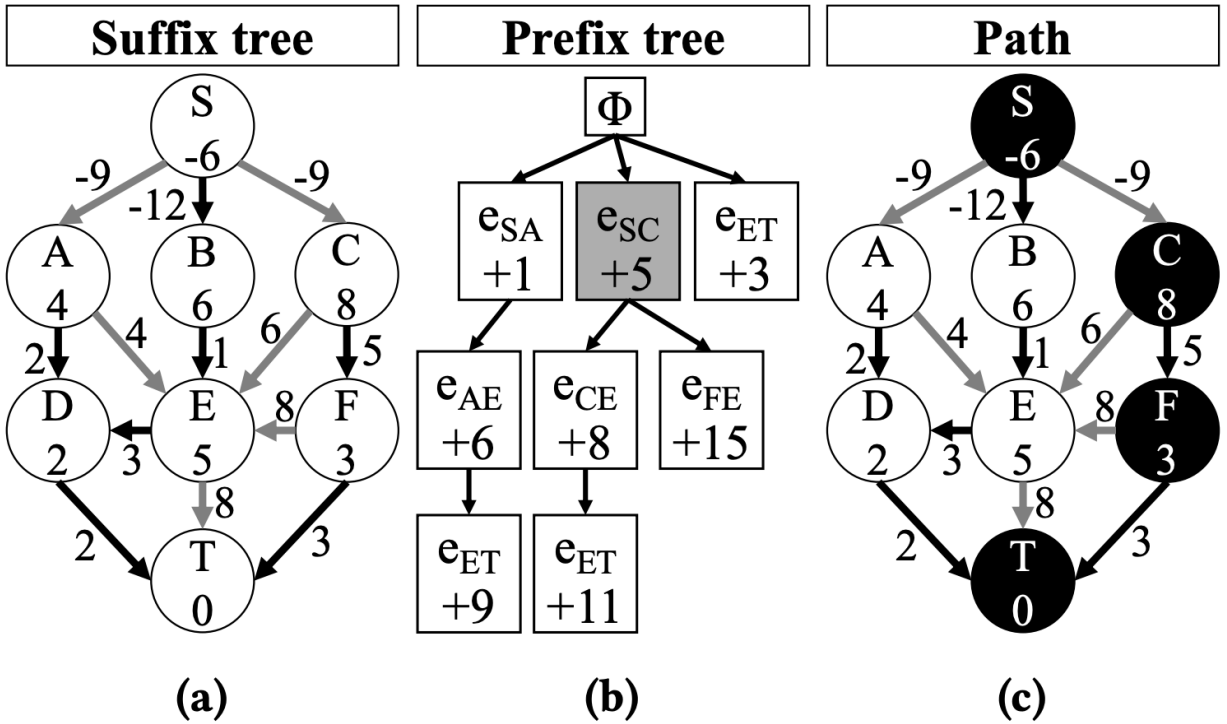
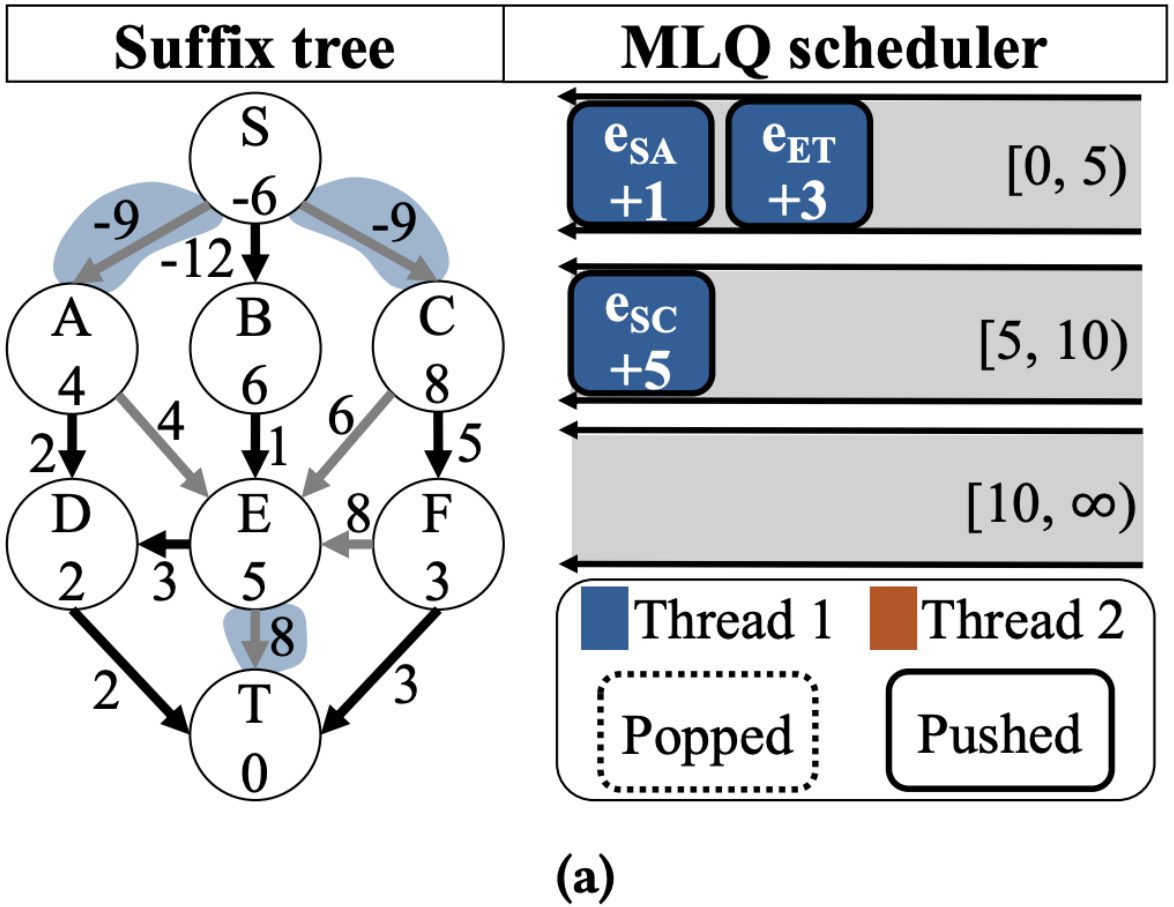


Fig. 1: Implicit path representation using suffix tree and prefix tree. Prefix $\langle e_{SC} \rangle$ + Suffix $\langle e_{CF}, e_{FT} \rangle$= Path $\langle e_{SC}, e_{CF}, e_{FT} \rangle$.

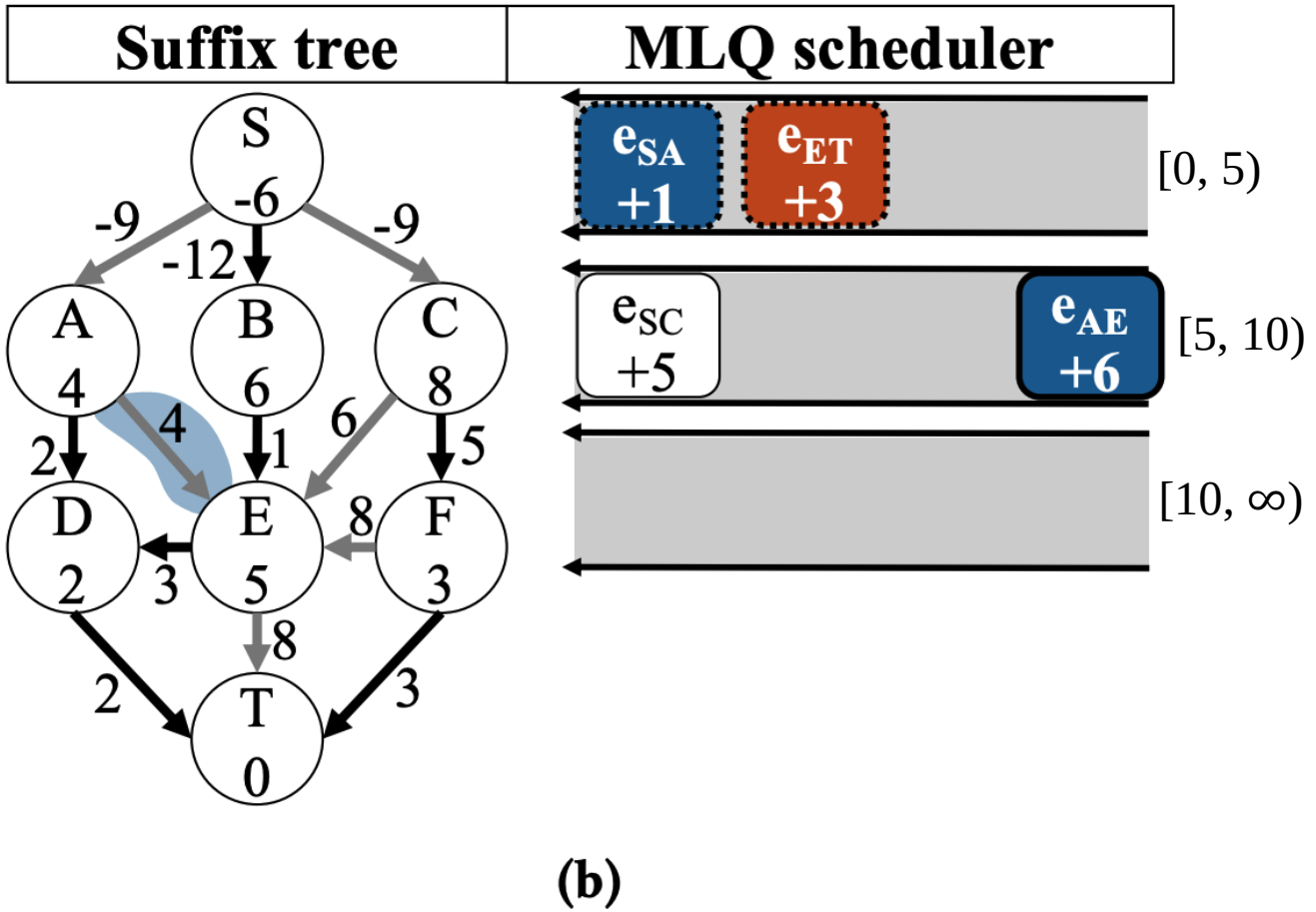
Multi-Level Queue Scheduler

- Example walkthrough
  - Three concurrent queues
  - Two threads
  - Slack range of each queue
    - [0, 5) → highest-priority
    - [5, 10)
    - [10, ∞) → lowest-priority
  - **Thread 1** generates e_{SA}, e_{ET}, e_{SC} and pushes them to their corresponding queue



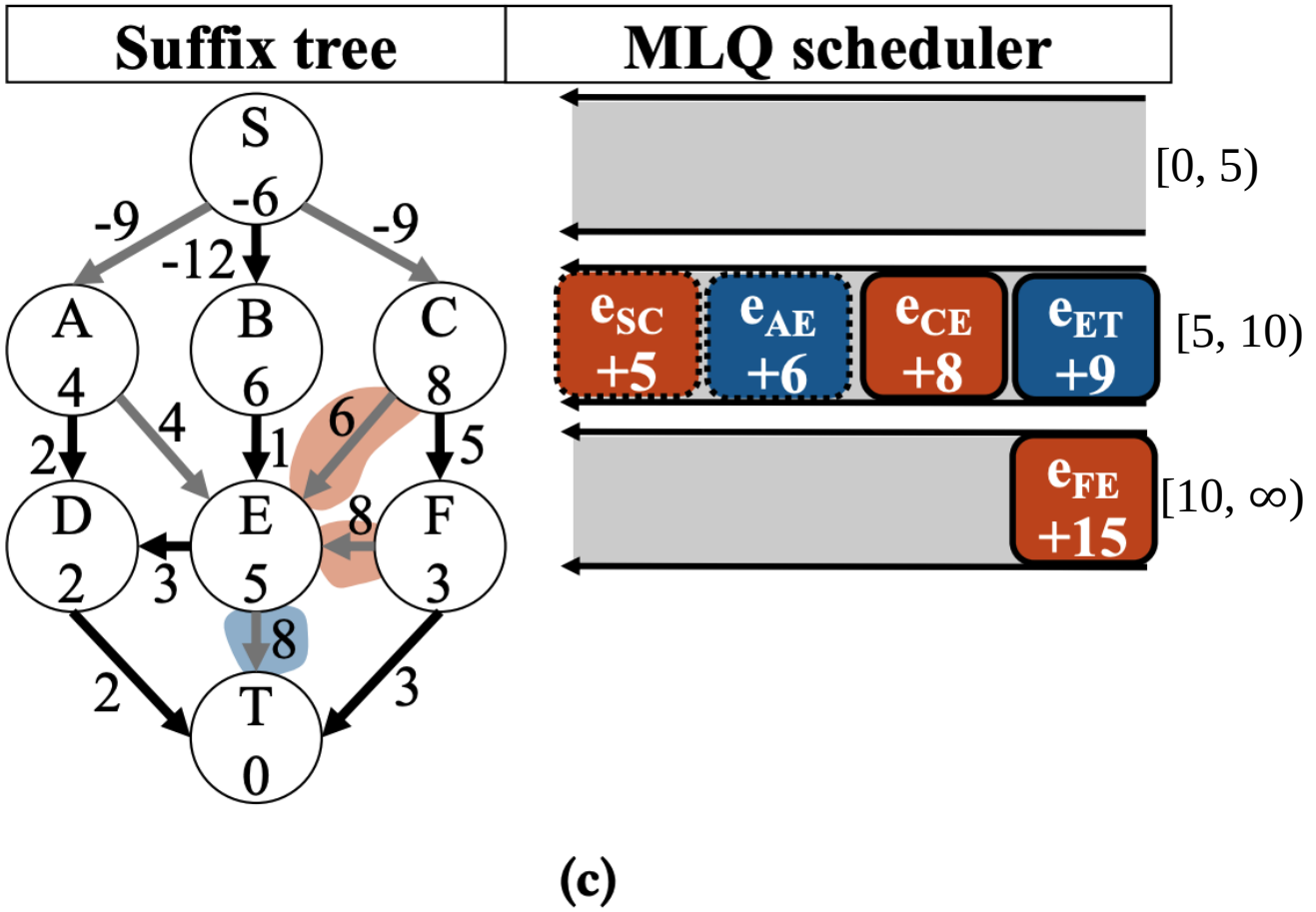
Multi-Level Queue Scheduler

- Example walkthrough
  - **Thread 1** pops e_{SA} and generates e_{AE}
  - **Thread 2** pops e_{ET} and generates nothing
  - Now slack range [0, 5) is empty, all the threads move onto the next queue



Multi-Level Queue Scheduler

- Example walkthrough
  - **Thread 2** pops e_{SC} and generates e_{CE} and e_{FE}
  - **Thread 1** pops e_{AE} and generates e_{ET}
  - All threads continues popping from slack range [5, 10) since it is not empty



Issue: Thread Contention

- Distribution of prefix tree nodes affects CPG performance
  - When path counts become unbalanced, certain queues experience high thread contention
  - Need to determine a good range for each queue to balance the path counts
- Real circuits exhibit **highly localized** slack distribution

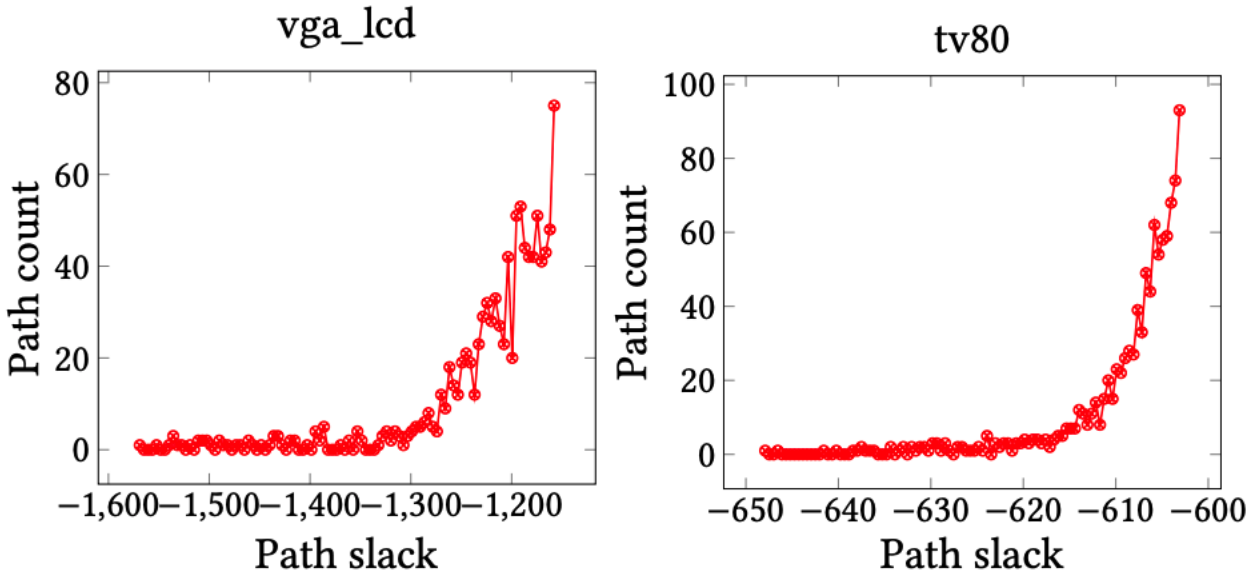
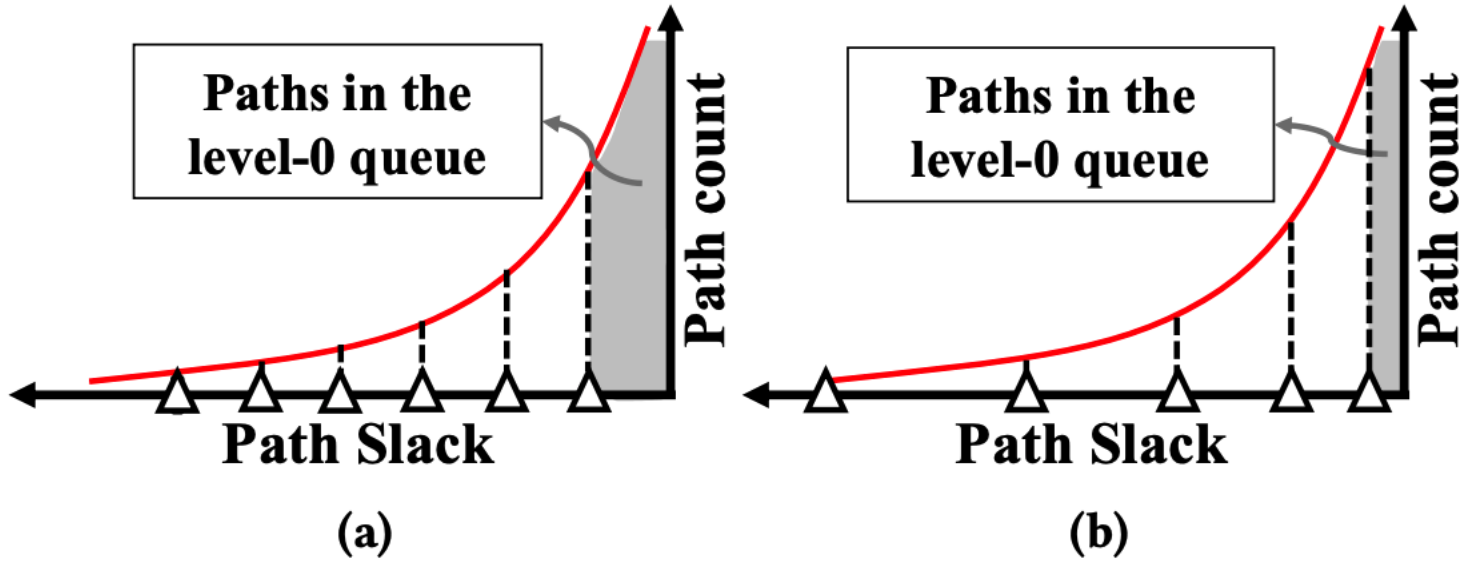


Fig. 4: Top-1K path slack distribution reported by OpenTimer [33].

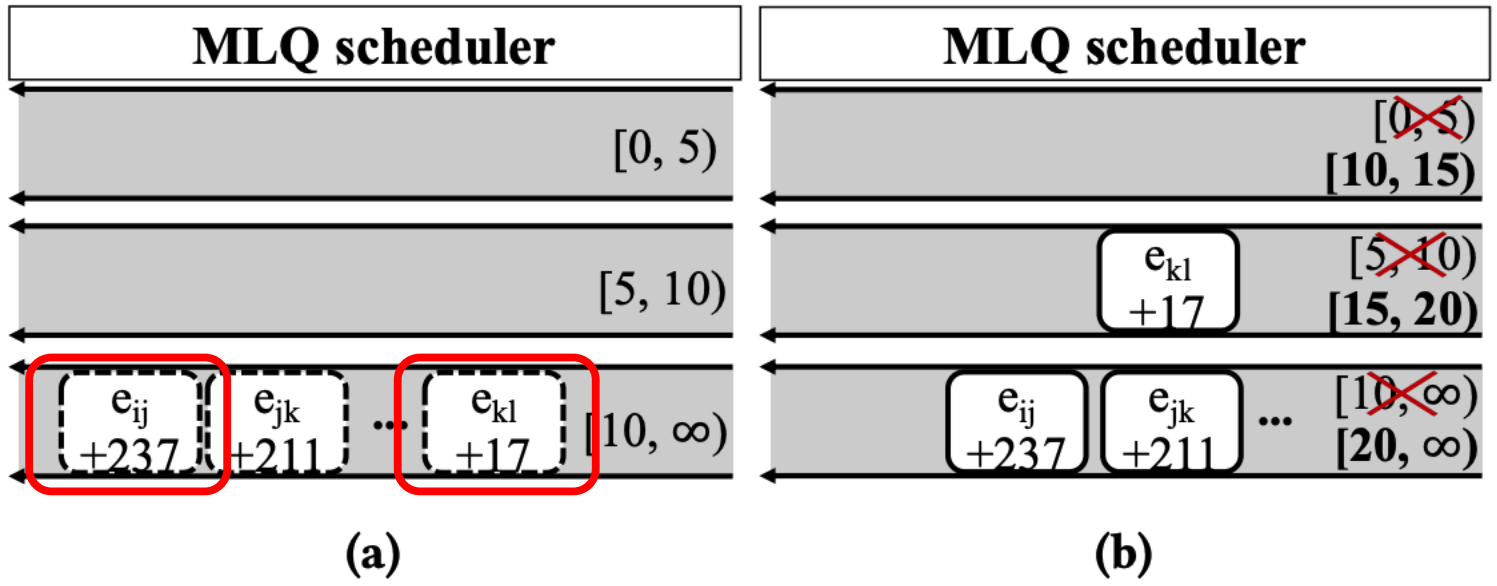
Balancing Path Counts in Each Queue

- We propose the geometric slack partitioning strategy
  - (a) sets equal ranges for each queue
    - The level-0 (highest-priority) queue manages significantly more paths than other queues
  - (b) sets ranges based on a **geometric sequence**
    - **More balanced** path count in each queue



Issue: Inaccurate Path Results

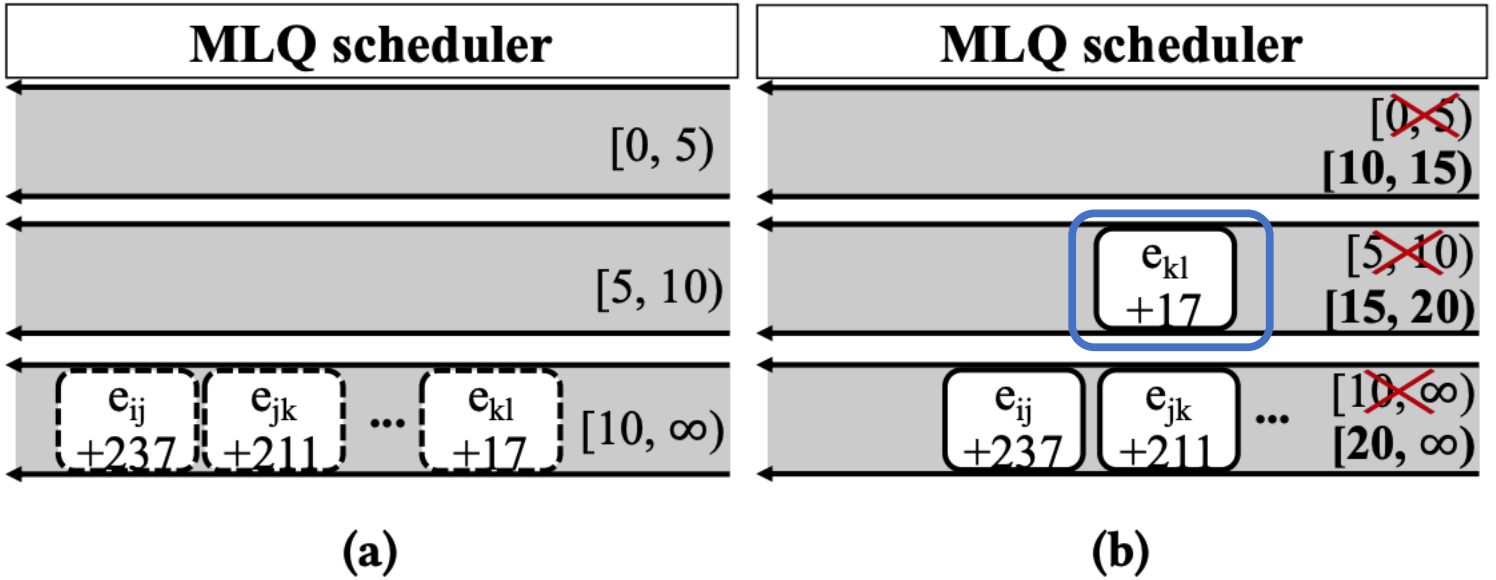
- CPG inaccuracy
  - The slack range of the lowest-priority queue is too big, resulting in a mix of high- and low-cost nodes
  - Path generation becomes inaccurate



+237 is processed before +17, inaccurate!

Reassigning Slack Priorities to Paths

- The node redistribution (NR) strategy
  - Reassigns priority to the nodes in the lowest-priority queue
  - Results in a more accurate processing order and higher accuracy



+17 is processed before +237, correct!

Experimental results

- PathGen is implemented in C++17 with optimization flag -O3
  - Using Taskflow[5] and Moodycamel[6] as parallel computing libraries
- Machine spec
  - CPU: 4.8-GHz Intel core i5-13500
  - OS: Ubuntu 22.04
- We compare PathGen with OpenTimer[3]
  - OpenTimer outperforms existing methods in both time and space complexities
  - With six circuit benchmarks generated from OpenTimer

[5]T. -W. Huang, D. -L. Lin, C. -X. Lin and Y. Lin, "Taskflow: A Lightweight Parallel and Heterogeneous Task Graph Computing System," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 6, pp. 1303-1320, 1 June 2022, doi: 10.1109/TPDS.2021.3104255.

[6]Moodycamel Concurrent Queue, 2014, https://github.com/cameron314/concurrentqueue

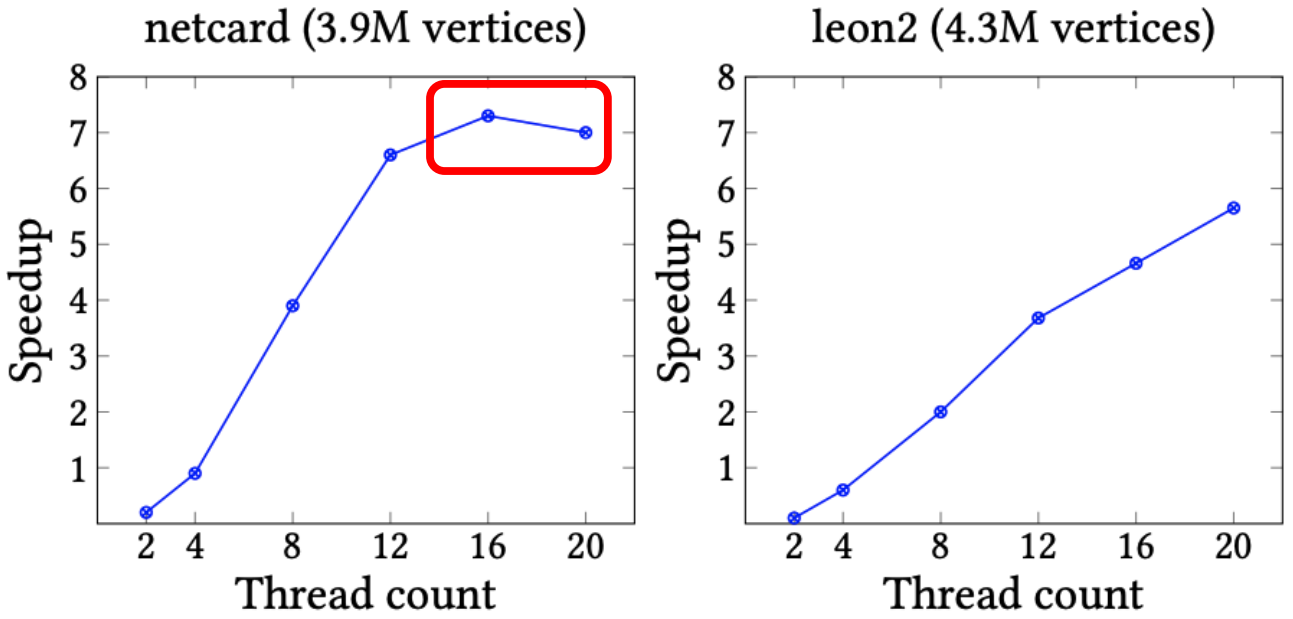
Overall Performance Comparison

- Comparison between OpenTimer and PathGen with 16 threads

				OpenTimer [33]		PathGen (16 threads)		
Circuit	$\|V\|$	$\|E\|$	Path count (K)	Runtime (ms)	Mem. (MB)	Runtime (ms)	Mem. (MB)	Avg. accuracy (%)
wb_dma	13124	16593	20	4.3	19.3	4.1 (**1.03**×)	24.5	99.9
des_perf	303690	387291	500	341.6	347.9	122.3 (**2.7**×)	461.3	100
vga_lcd	397816	498873	1000	1076.5	552.4	401.9 (**2.6**×)	954.5	100
leon3mp	3376842	4148798	1000	2243.1	3261.7	400.5 (**5.6**×)	4540.9	100
netcard	3999174	4903397	1000	2135.8	3574.6	292.5 (**7.3**×)	4199.7	100
leon2	4328285	5273106	1000	2552.8	4065.4	344.9 (**7.4**×)	5774.3	100

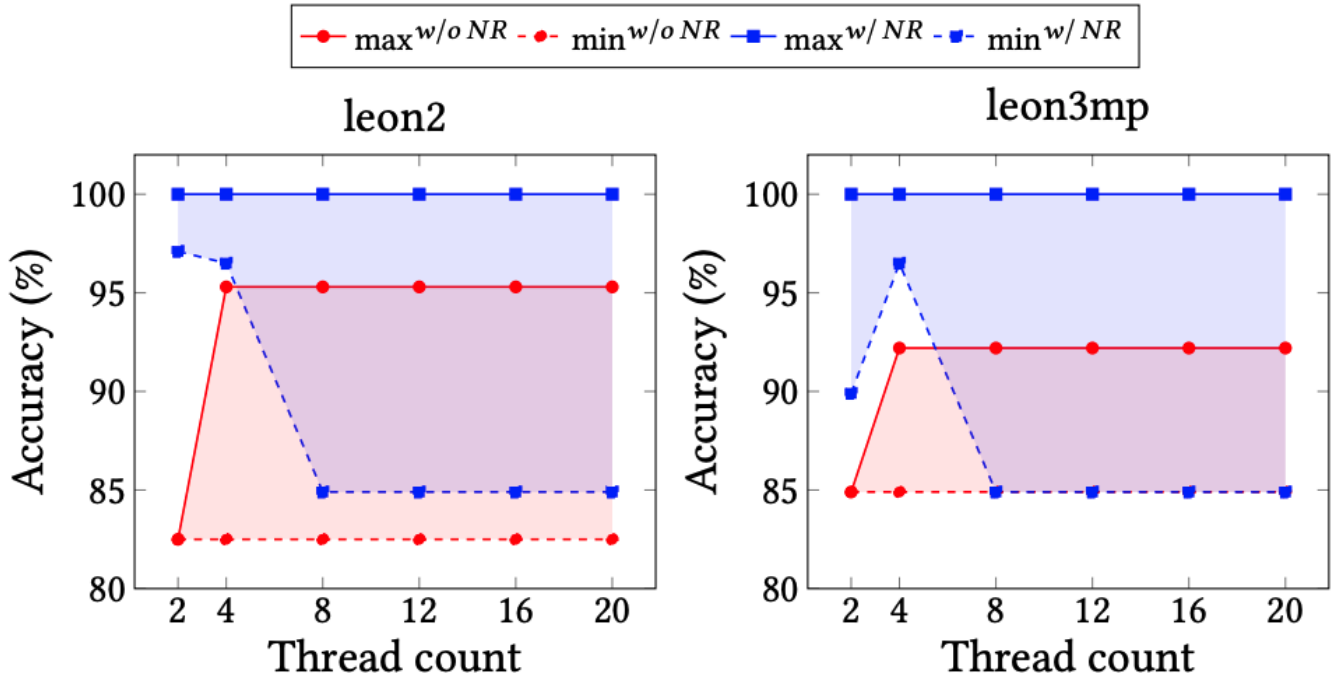
Speedup of PathGen over OpenTimer

- Speedup of PathGen over OpenTimer at different thread count
  - The more threads we use, the faster we can clear a queue and generate paths
  - Using the maximum thread count does not always yield the optimal performance
    - Thread contention can slow down queue access



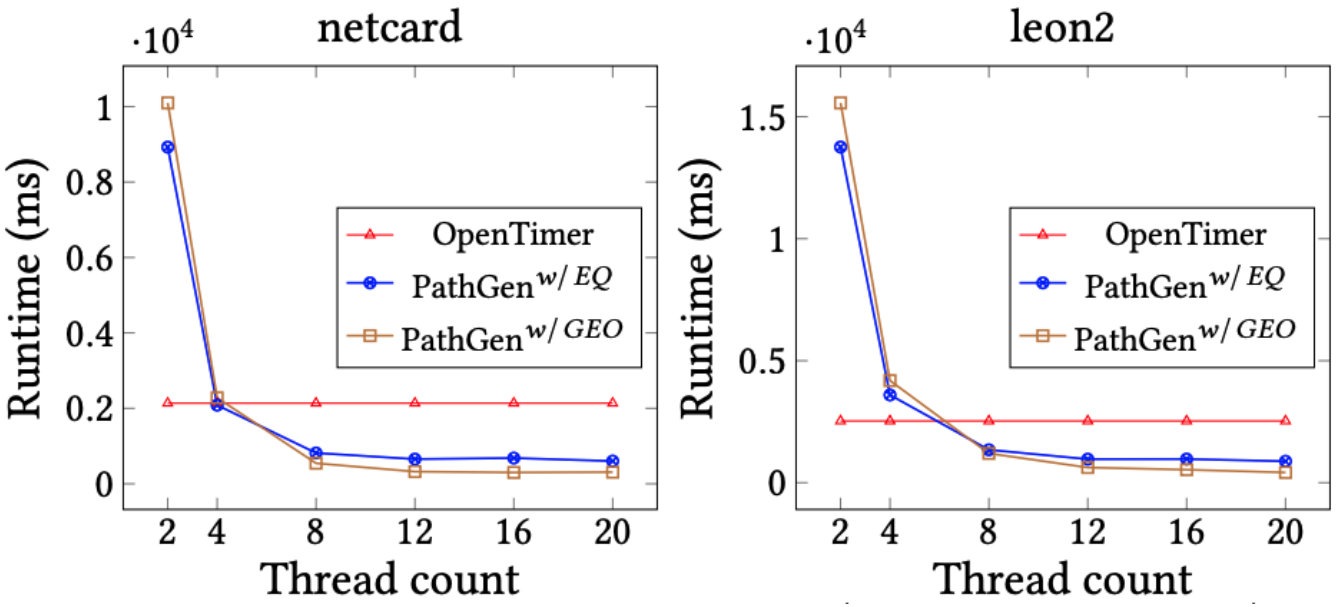
Accuracy of PathGen

- Accuracy of PathGen w/ NR and w/o NR at different thread counts
  - The accuracy of PathGen w/o NR ranges from 82-95%
  - The accuracy of PathGen w/ NR ranges from 86-100%
    - 4%/5% improvement on min/max accuracy



Runtime of Different Slack Partitioning Strategies

- Runtime of OpenTimer, PathGen$^{w/EQ}$ and PathGen$^{w/GEO}$ at different thread counts
  - In leon2, PathGen$^{w/GEO}$ is 1.5×, 2×, and 2.2× faster than PathGen$^{w/EQ}$ at eight threads, 12 threads, and 16 threads
  - Demonstrates the effectiveness of geometric slack partitioning



EQ: equal ranges
GEO: geometric ranges

Conclusion

- We have introduced PathGen
  - Efficiently groups paths into multiple queues of different slack priorities to run in parallel
  - Balances the number of paths in each queue to minimize contention
  - Transfers the paths between queues to adjust slack priorities to process paths in a more accurate order


WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON