OpenTimer: A high-performance timing analysis tool

LibreCores Student Design Contest

Tsung-Wei Huang, Chun-Xun Lin, and Martin D. F. Wong

Department of Electrical and Computer Engineering (ECE)

University of Illinois at Urbana-Champaign (UIUC), IL, USA

FOSSi Foundation

ORCONF2016

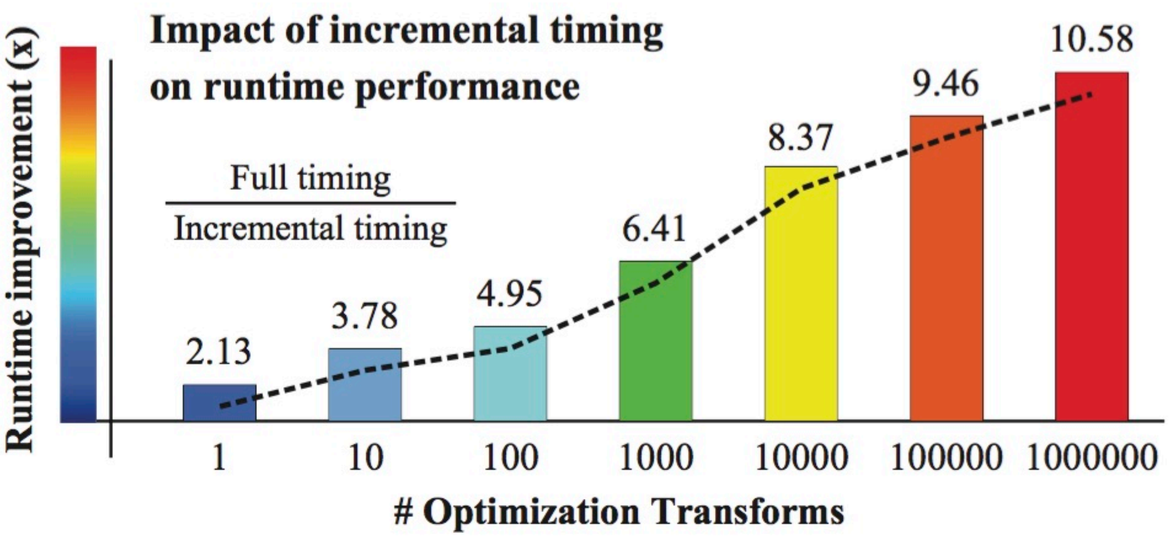
ECE ILLINOIS

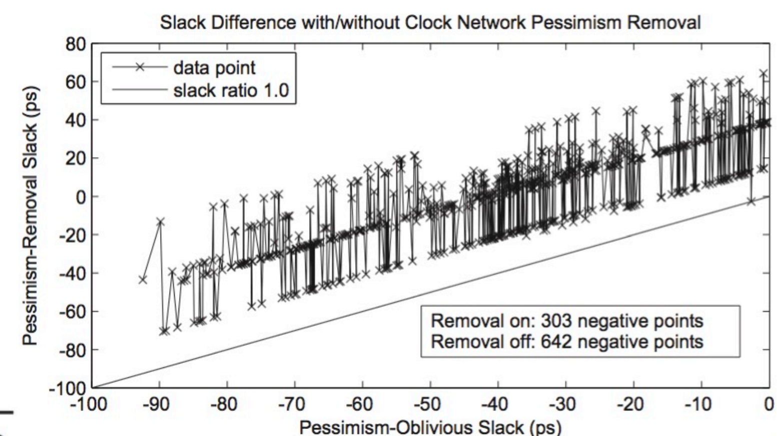
ILLINOIS

Motivation of OpenTimer Project – TAU Contests

- An open-source STA engine with incremental timing and CPPR
  - Important for timing-driven applications
  - Fast full timing and incremental timing analysis
  - Capability of path-based CPPR analysis
  - Parallel programming and multi-threading

Prestigious 2014-2016
TAU timing contests



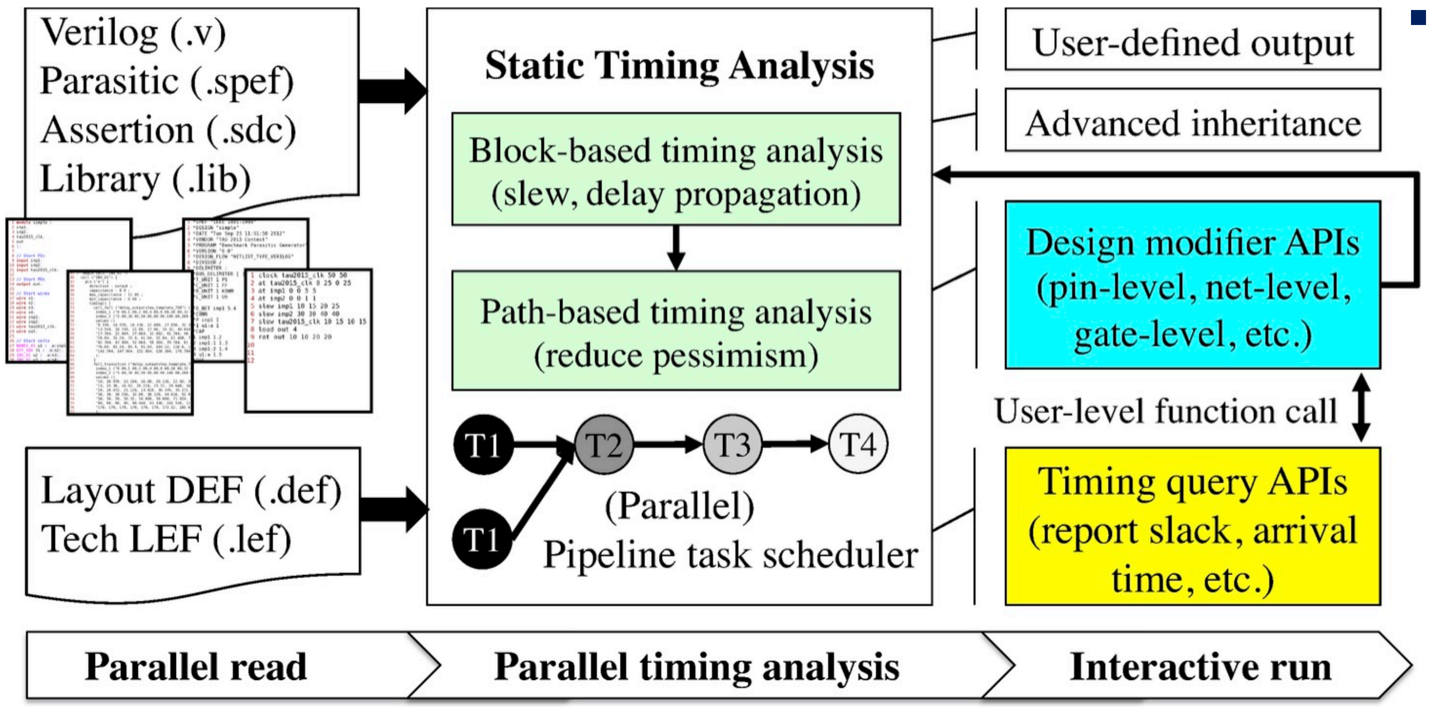


CPPR impact
(reduction of unwanted pessimism)

*CPPR stands for Common Path Pessimism Removal

OpenTimer Architecture

- An open-source high-performance timing analysis tool
  - TAU14 (1st place), TAU15 (2nd place), TAU16 (1st place)
  - Selected as the golden timer in ICCAD15, TAU16, and TAU17 contests



- Feature highlights
  - C++11
  - Industry format
  - STA engine
  - Block-based
  - Path-based
  - Incremental
  - Lazy evaluation
  - CPPR
  - Multi-threaded

http://web.engr.illinois.edu/~thuang19/software/timer/OpenTimer.html

Experimental Results – Overall Performance Comparison

TABLE I

PERFORMANCE COMPARISON BETWEEN OPENTIMER AND TOP-RANKED TIMERS IITRACE AND ITIMERC 2.0 FROM TAU 2015 CAD CONTEST [1].

Circuit	#Gates	#Nets	#OPs	iitRACE			iTimerC 2.0			OpenTimer		
				accuracy	runtime	memory	accuracy	runtime	memory	accuracy	runtime	memory
b19	255.3K	255.3K	5641.5K	63.03 %	629 s	3.0 GB	99.95 %	215 s	5.8 GB	99.95 %	52 s	4.6 GB
cordic	45.4K	45.4K	1607.6K	61.83 %	100 s	0.9 GB	98.88 %	80 s	1.3 GB	98.88 %	18 s	1.3 GB
des_perf	138.9K	139.1K	4326.7K	67.43 %	299 s	4.2 GB	97.02 %	92 s	3.1 GB	99.73 %	30 s	3.0 GB
edit_dist	147.6K	150.2K	3368.3K	64.83 %	857 s	2.0 GB	98.29 %	98 s	3.8 GB	98.30 %	42 s	3.8 GB
fft	38.2K	39.2K	1751.7K	89.66 %	70 s	0.5 GB	98.45 %	49 s	1.2 GB	99.77 %	11 s	1.2 GB
leon2	1616.4K	1517.0K	8438.5K	72.34 %	16832 s	9.9 GB	100.00 %	787 s	27.2 GB	100.00 %	282 s	22.8 GB
leon3mp	1247.7K	1248.0K	8405.9K	62.99 %	4960 s	8.2 GB	100.00 %	609 s	19.8 GB	100.00 %	163 s	17.9 GB
mgc_edit_dist	161.7K	164.2K	3403.4K	64.29 %	1578 s	1.9 GB	100.00 %	135 s	4.1 GB	100.00 %	41 s	3.1 GB
mgc_matrix_mult	171.3K	174.5K	3717.5K	67.93 %	1363 s	2.0 GB	100.00 %	157 s	4.3 GB	100.00 %	31 s	3.1 GB
netcard	1496.0K	1497.8K	11594.6K	87.63 %	6662 s	9.4 GB	99.99 %	691 s	22.9 GB	99.99 %	192 s	20.8 GB
cordic_core	3.6K	3.6K	226.0K	59.42 %	21 s	0.3 GB	95.19 %	29 s	0.2 GB	95.19 %	3 s	0.1 GB
crc32d16N	478	495	28.9K	57.15 %	3 s	0.1 GB	100.00 %	5 s	0.1 GB	100.00 %	1 s	0.1 GB
softusb_navre	6.9K	7.0K	427.8K	40.17 %	21 s	0.1 GB	0.00 %	-	-	99.97 %	4 s	0.5 GB
tip_master	37.7K	38.5K	1300.4K	82.95 %	64 s	0.6 GB	96.42 %	47 s	1.0 GB	97.04 %	9 s	0.8 GB
vga_lcd_1	139.5K	139.6K	2961.5K	99.65 %	260 s	1.6 GB	100.00 %	94 s	2.2 GB	100.00 %	31 s	2.9 GB
vga_lcd_2	259.1K	259.1K	12674.7K	98.57 %	1132 s	13.3 GB	100.00 %	156 s	5.0 GB	100.00 %	65 s	3.9 GB

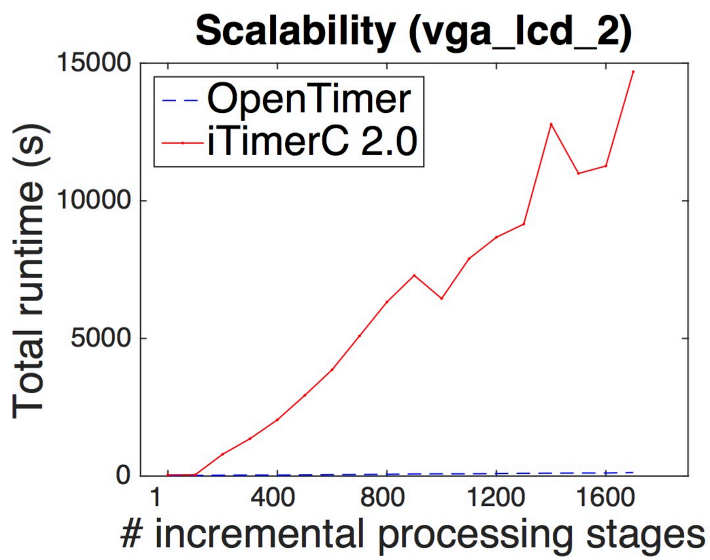
#Gates: number of gates. #Nets: number of nets. #OPs: number of operations. accuracy: average of path accuracy and value accuracy (%). -: program crash.

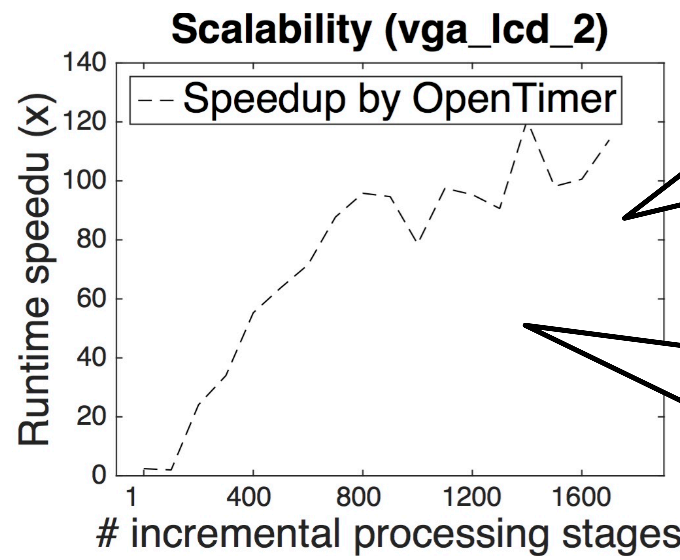
**iTimerC 2.0: IEEE/ACM ICCAD15 (binary from authors)*

**iitRACE: IEEE/ACM ICCAD15 (binary from authors)*

Golden reference by IBM Einstimer

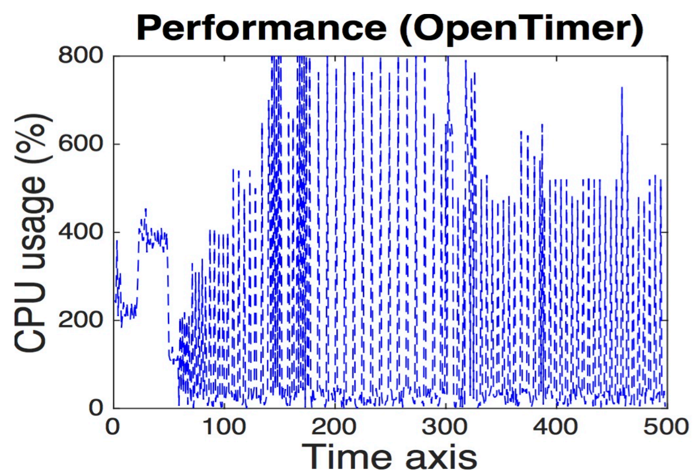
Experimental Results – Scalability of Incremental Timing

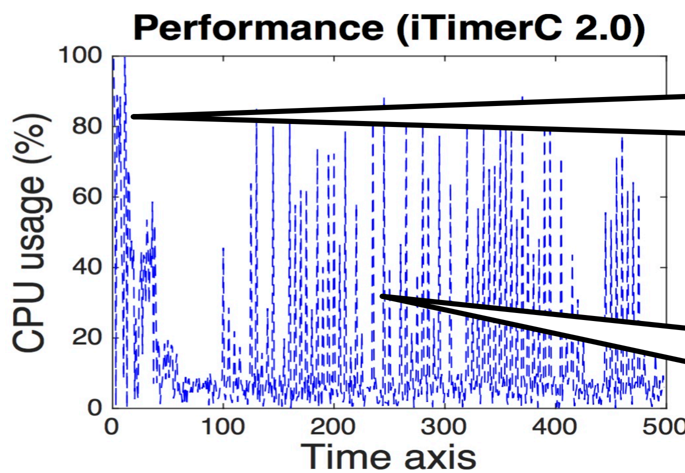




x115 speedup by OpenTimer (at 1459th stage)

X2.7 speedup by OpenTimer (at 1th stage, i.e. full timing)





Parallel IO by OpenTimer

Parallel timing by OpenTimer

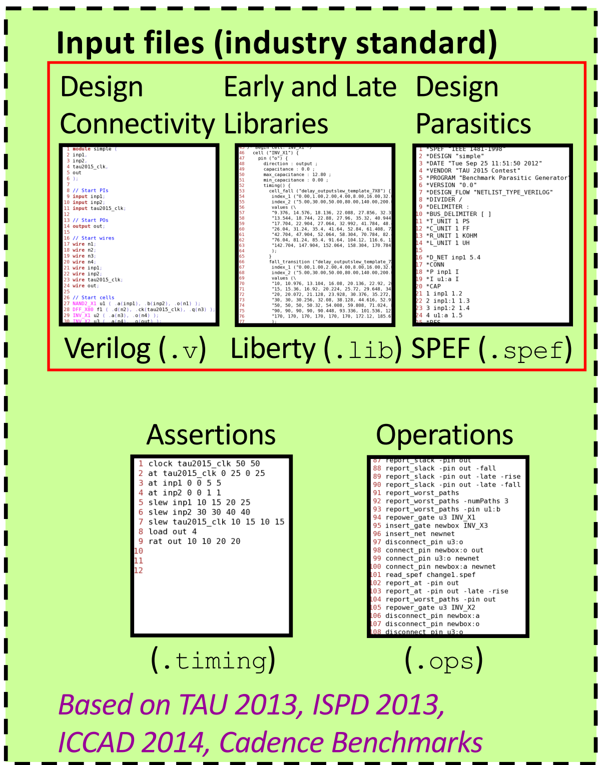
Conclusion

- Developed a high-performance timing analysis tool
  - Free software and open-source under GPL v3.0
  - Industry format (.v, .spef, .lib, .lef, .def, etc.)
  - Fast, accurate, robust, and CPPR by default
- Recognitions and contributions to community
  - 1st prize in TAU14 contest (full timing with CPPR)
  - 2nd prize in TAU15 contest (incremental timing with CPPR)
  - 1st prize in TAU16 contest (timing macro-modeling)
  - Golden timer in ICCAD15 CAD contest
  - Golden timer in TAU16 contest
  - Golden timer in TAU17 contest
- Acknowledgment
  - Jin, Billy, M.-C., team iTimerC, team iitRACE, and the UIUC CAD group!

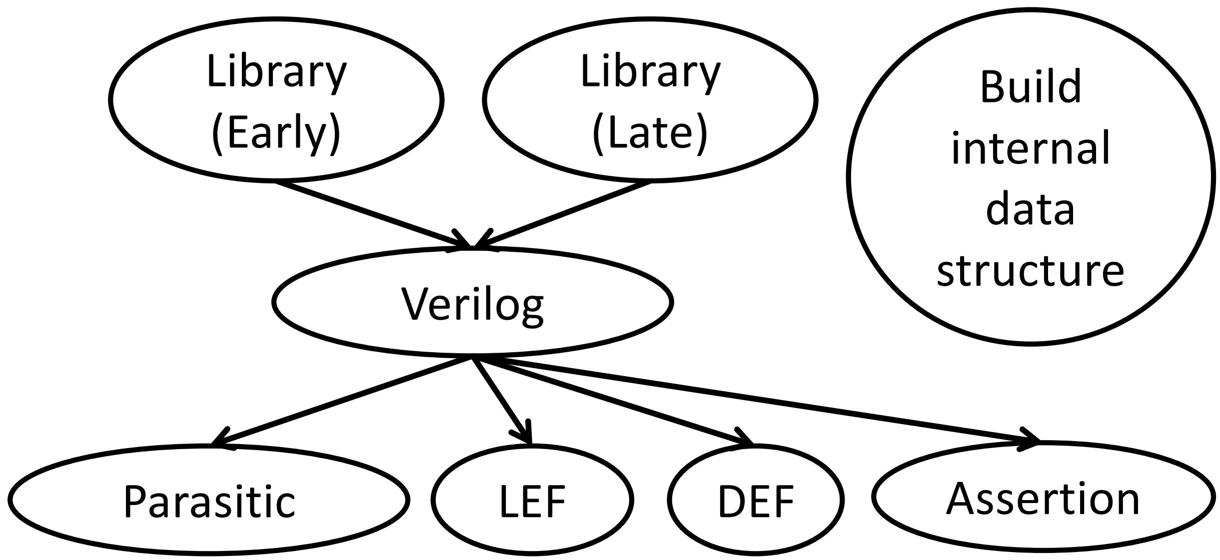
Backup slides

Initialize the Timer – Parallel IO

- A set of design files that follow the industry standard format
  - Verilog netlist, two libraries (early and late), parasitic spef, etc.
  - Time-consuming IO (e.g., file IO, complex parsing)



Task graph



Parallel dependency generation using portable OpenMP

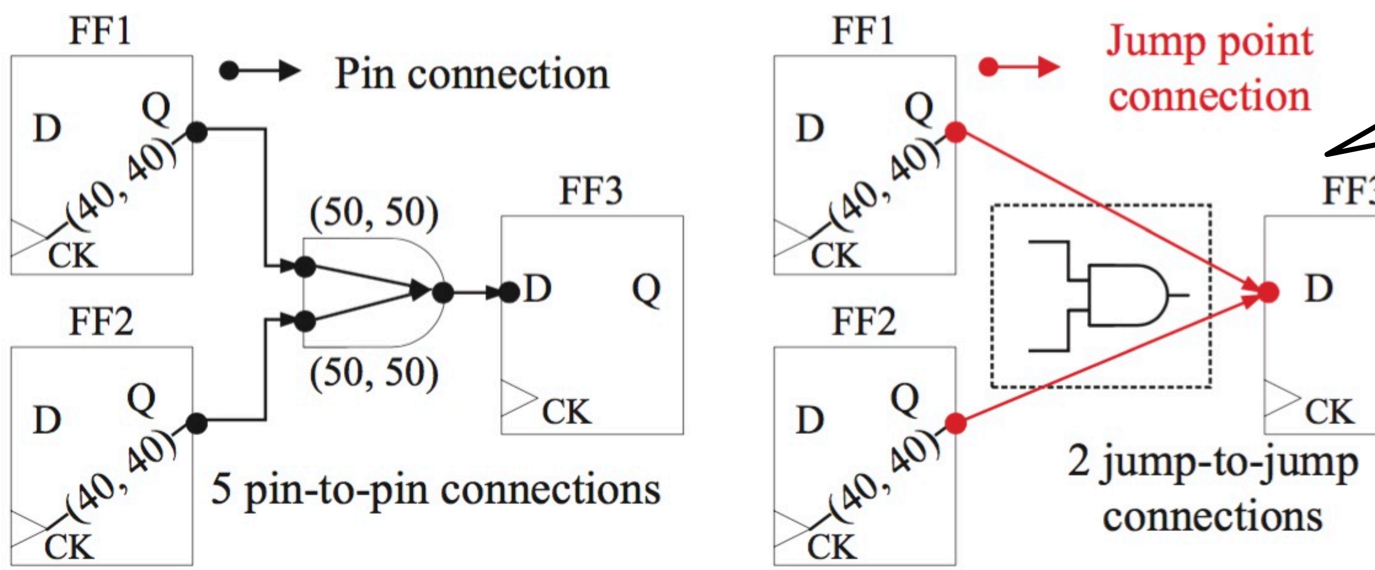
#pragma parallel …

#pragma task …

x2 speedup by parallel read/parse!

Timing Graph Reduction

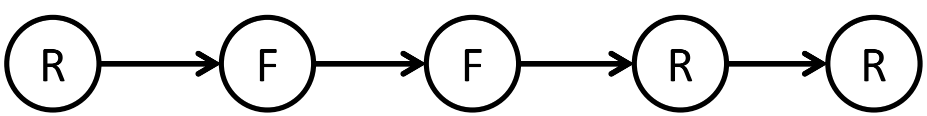
- Reduce the search space
  - Identify tree-structured subgraphs in the original timing graph
  - Merge every leaf-root path (transition-definite)



(a) Ordinary circuit graph (b) Contracted circuit graph

~30% reduction on the graph size

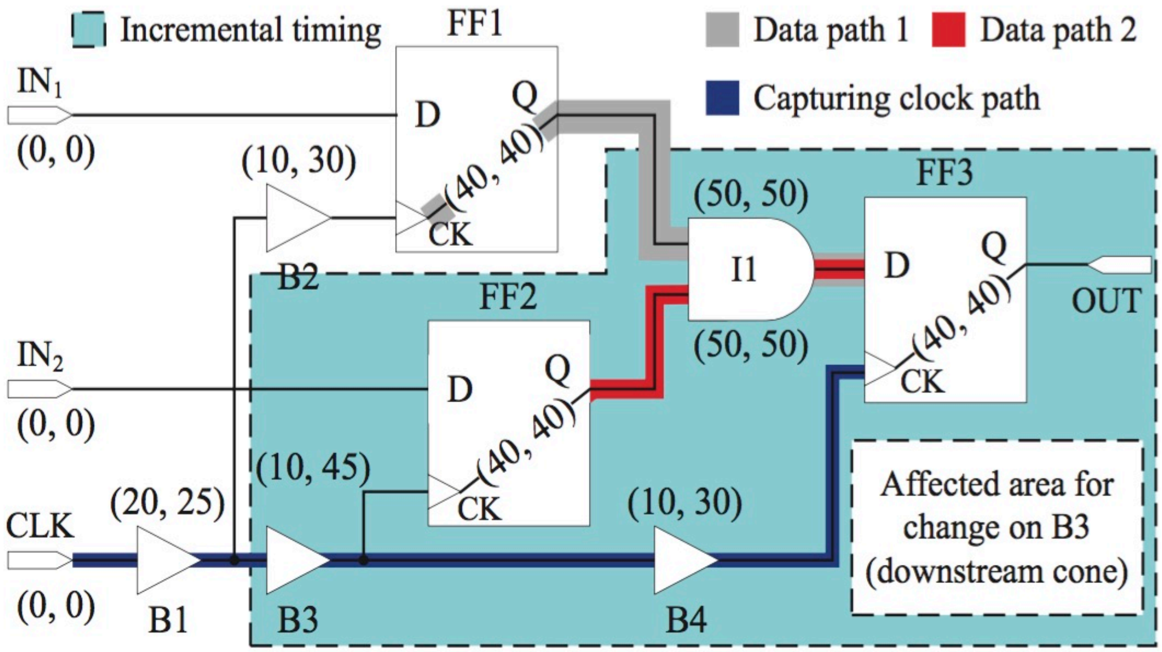
Apply to combinational circuits only



Every leaf-root path can be uniquely defined (given a transition at an endpoint)

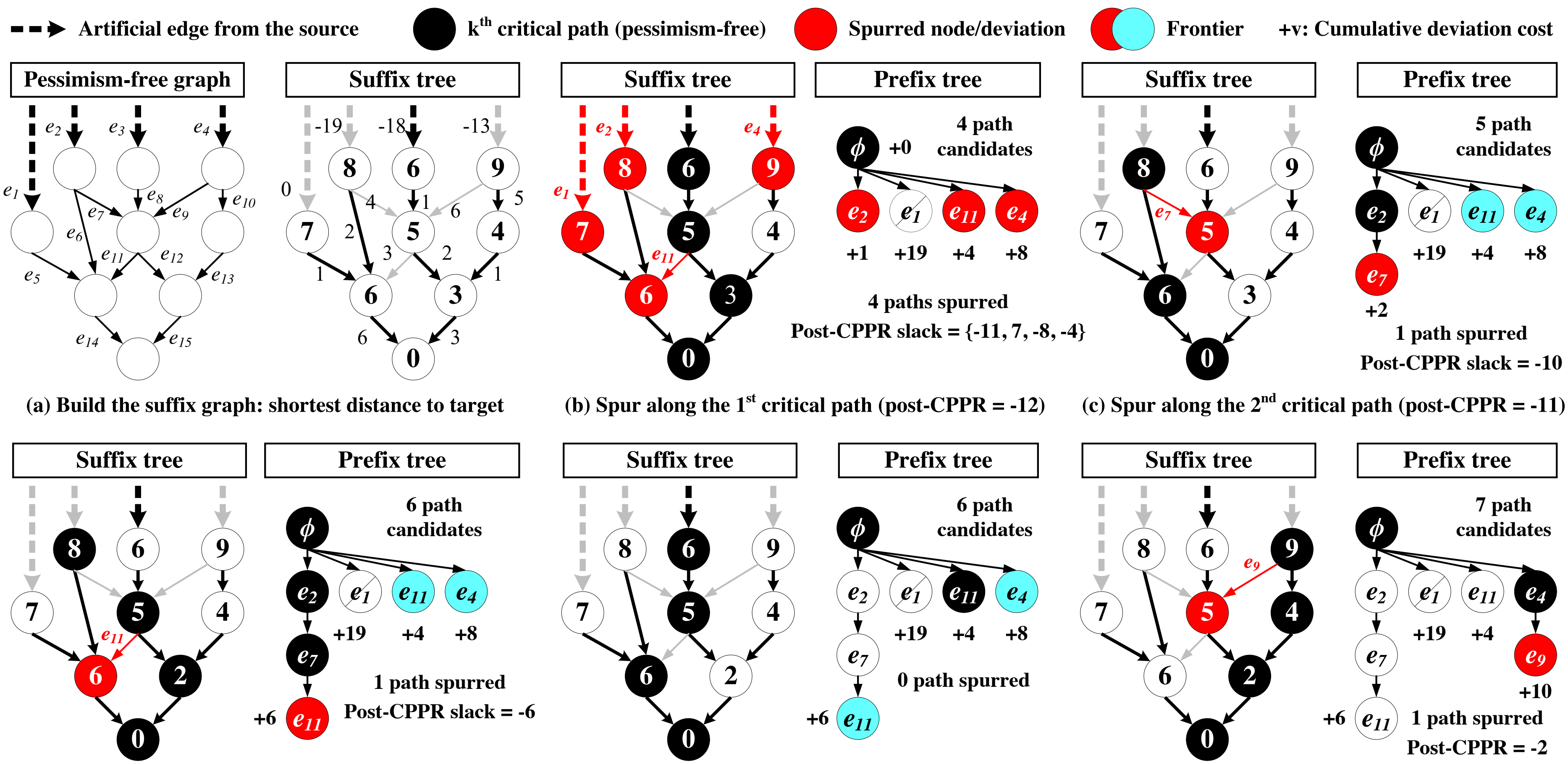
Key Components of Incremental Timing

- Full timing is just a special case of incremental timing
- Design modifiers
  - Pin-level operations, net-level operations, and gate-level operations
- Timing queries
  - Slack
  - Arrival time
  - Required time
  - TNS and WNS
  - Critical path report
  - CPPR
- Source of propagation
- Lazy evaluation
- Explore parallel incremental timing



Common path pessimism removal (CPPR)

- Constant-space and -time representation for CPPR



(a) Build the suffix graph: shortest distance to target

(b) Spur along the 1st critical path (post-CPPR = -12)

(c) Spur along the 2nd critical path (post-CPPR = -11)

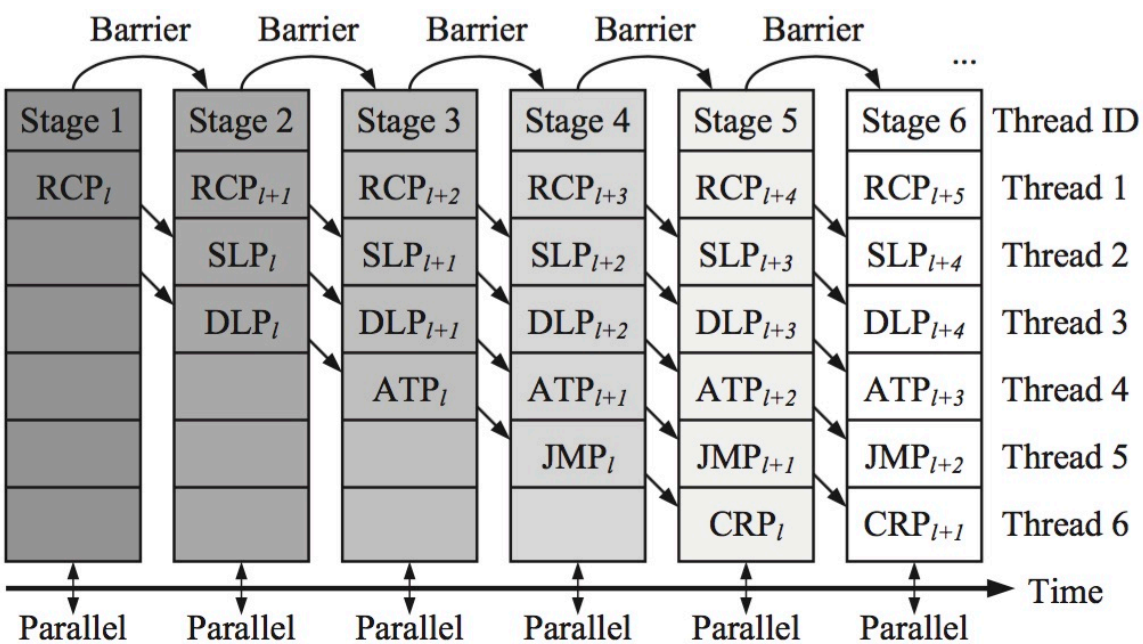
(d) Spur along the 3rd critical path (post-CPPR = -10)

(e) Spur along the 4th critical path (post-CPPR = -8)

(f) Spur along the 6th critical path (post-CPPR = -4)

Pipeline-based Parallel Timing Propagation

- Timing propagation has several linearly dependent tasks
  - RC update → Slew & Delay → Arrival time → Jump point → CPPR
  - Pipeline scheduling with multiple threads



Algorithm 18: update_timing()

```
B ← bucket list of the timer;
if B.num_pins = 0 then
    return;
end
IncrementalLevelization(B);
l_min ← B.min_nonempty_level;
l_max ← B.max_nonempty_level;
# Parallel_Region {
# Master_Thread_do for l = l_min to l_max + 4 do
    # Fork_Thread_Task PropagateRC(l);
    # Fork_Thread_Task PropagateSlew(l − 1);
    # Fork_Thread_Task PropagateDelay(l − 1);
    # Fork_Thread_Task PropagateArrivalTime(l − 2);
    # Fork_Thread_Task PropagateJumpPoint(l − 3);
    # Fork_Thread_Task PropagateCPPRCredit(l − 4);
    # Synchronize_Thread_Tasks;
end
};
# Parallel Region {
# Master_Thread_do for l = l_max to B.min_non_empty_level do
    # Fork_Thread_Task PropagateFanin(l);
    # Fork_Thread_Task PropagateRequiredArrivalTime(l);
    # Synchronize_Thread_Tasks;
end
};
remove all pins from the bucket list B;
```

We use the following paper for dealing with CPPR

**UI-Timer: An ultra-fast clock network pessimism removal algorithm, T.-W. Huang, P.-C. Wu, and Martin D. F. Wong, ICCAD14*