ParallelClosure: A Parallel Design Optimizer for Timing Closure

Yi-Shan Lu[1], Wenmian Hua[2], Rajit Manohar[2], Keshav Pingali[1]
[1]University of Texas at Austin, [2]Yale University

March 22nd, 2019 at TAU 2019 Workshop

ParallelClosure

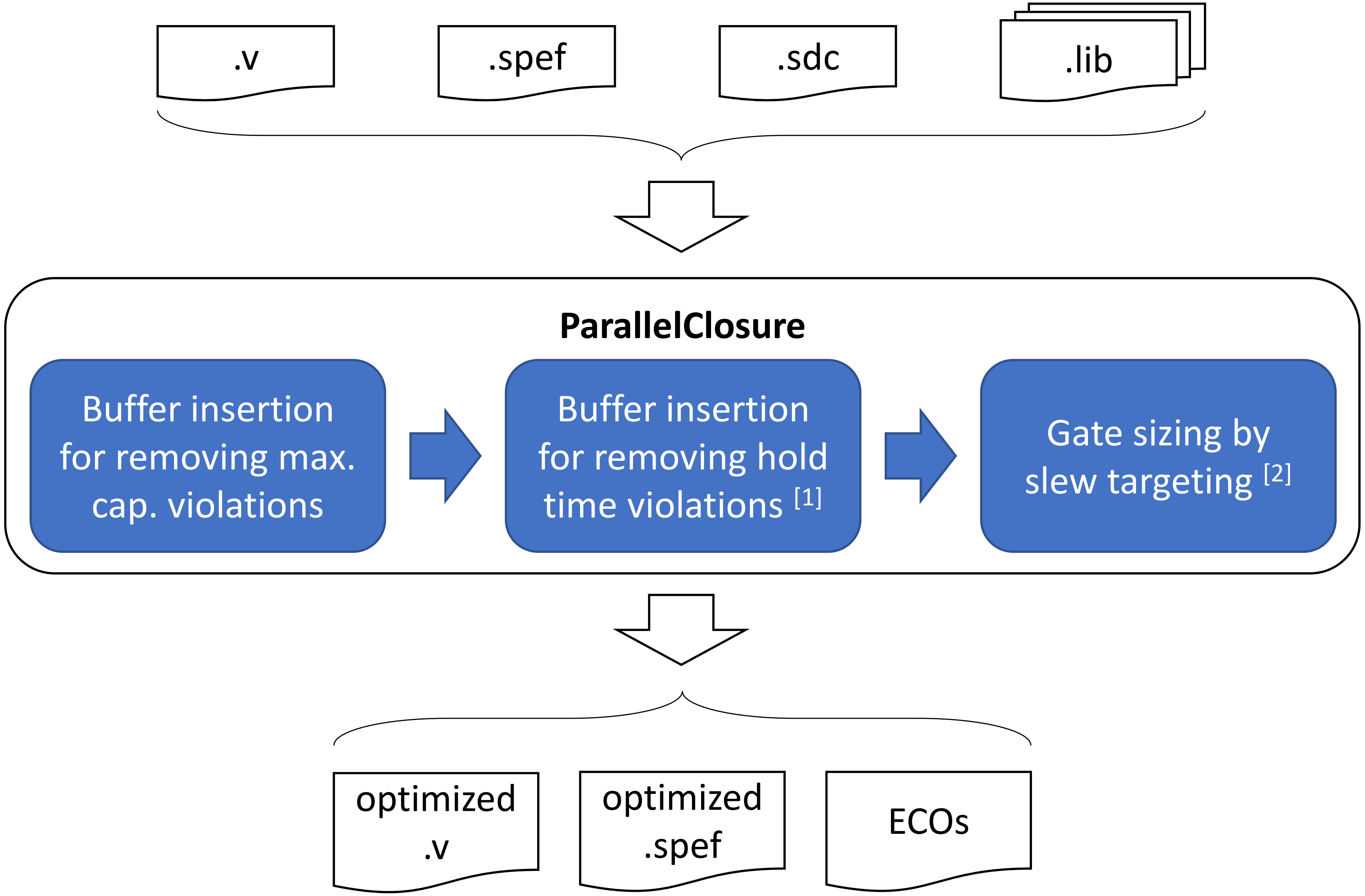
1. N. V. Shenoy, R. K. Brayton, A. L. Sangiovanni-Vincentelli. "Minimum padding to satisfy short path constraints," in *ICCAD*'93.
2. S. Held. "Gate sizing for large cell-based designs," in *DATE*'09.
3. K. Pingali et al. "The TAO of parallelism in algorithms," in *PLDI*'11.
4. D. Nguyen, A. Lenharth, K. Pingali. "A lightweight infrastructure for graph analytics," in *SOSP*'13.

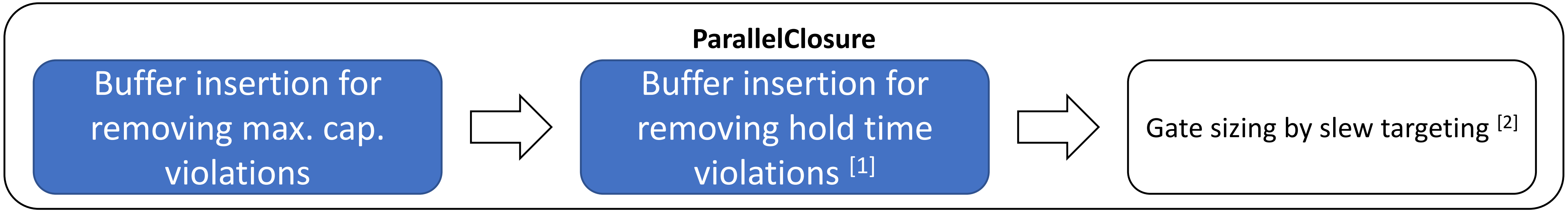
- Our design optimizer for TAU 2019 contest
- Design optimizations considered
  - Buffer insertion for fixing hold time violations [1]
  - Gate sizing by slew targeting [2] for minimizing area, leakage power & clock period
  - All algorithms are generalized for multi-corner, multi-mode (MCMM) optimizations
- Parallelization of static timing analysis (STA) & gate sizing
  - Parallelism analyses using the operator formulation [3]
  - Parallel implementation using the shared-memory Galois framework [4]

Outline

- Optimization flow – the algorithms
- Parallelization – boosting tool runtime
- Limitation
- Conclusions

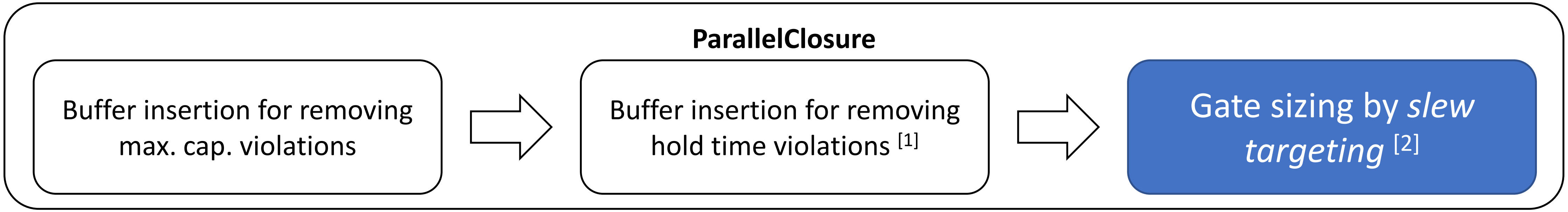
1. N. V. Shenoy, R. K. Brayton, A. L. Sangiovanni-Vincentelli. "Minimum padding to satisfy short path constraints," in *ICCAD*'93.
2. S. Held. "Gate sizing for large cell-based designs," in *DATE*'09.





- We generalize the approach in the following paper to MCMM:

  [1] N. V. Shenoy, R. K. Brayton, A. L. Sangiovanni-Vincentelli. “Minimum padding to satisfy short path constraints,” in *ICCAD*’93. (UC Berkeley CAD group)



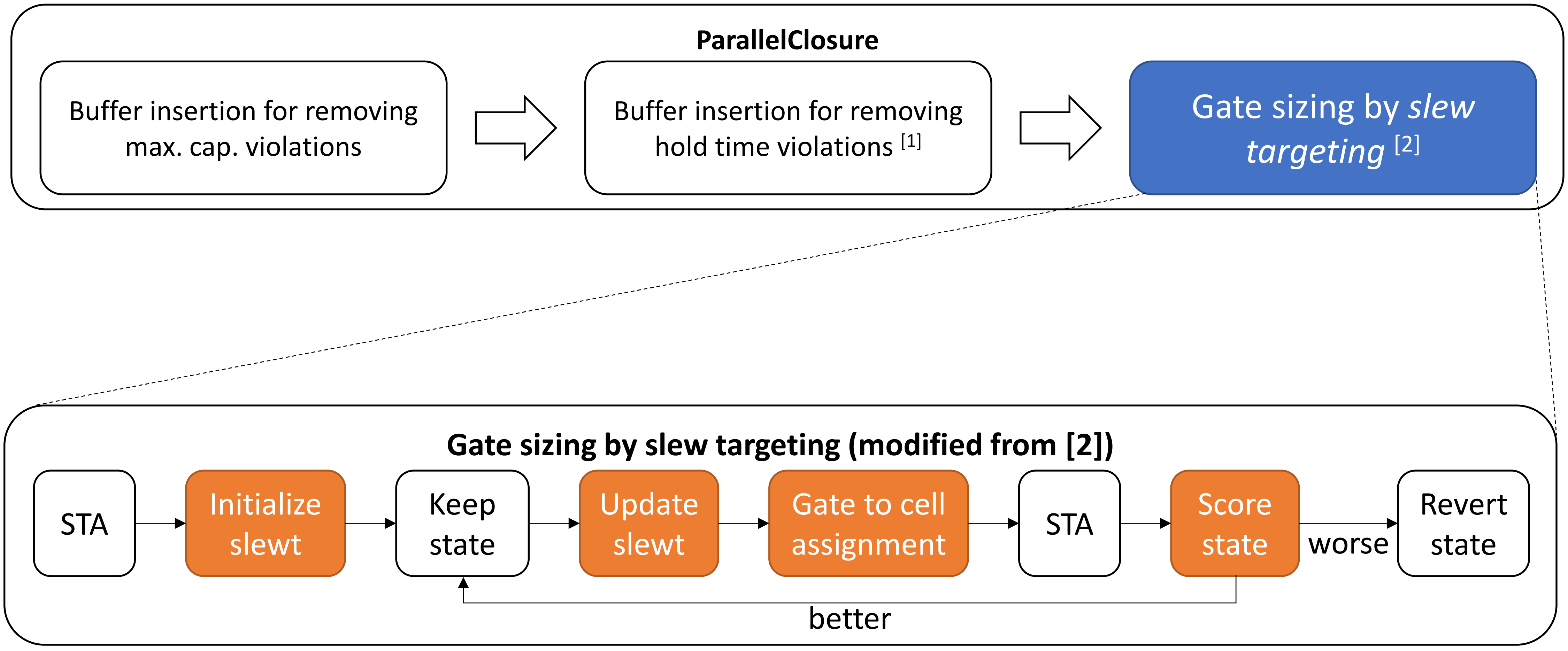
- Gate sizing in multi-mode optimization

Gate position	Setup time	Hold time
On critical paths	Upsize	Downsize
Not on critical paths	Downsize	Upsize

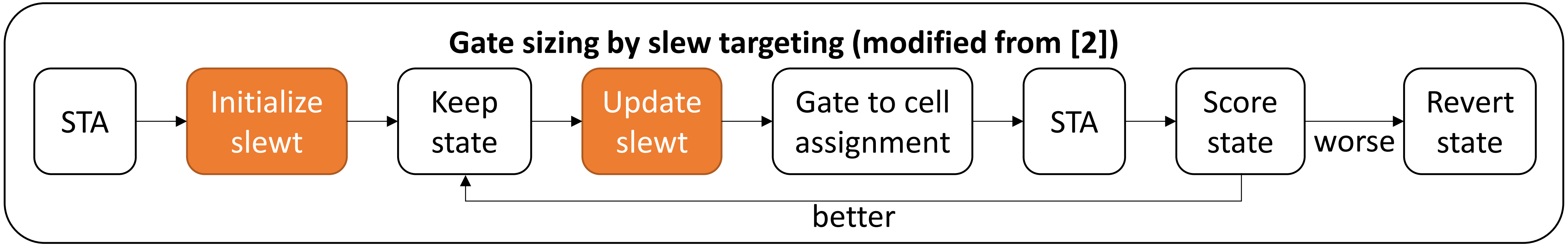
- Each gate output has a slew target per combination of (corner, mode)
- Use slew targets (slewt) to guide the sizing process

Sizing operation	Slew target
Upsize	Decrease
Downsize	Increase

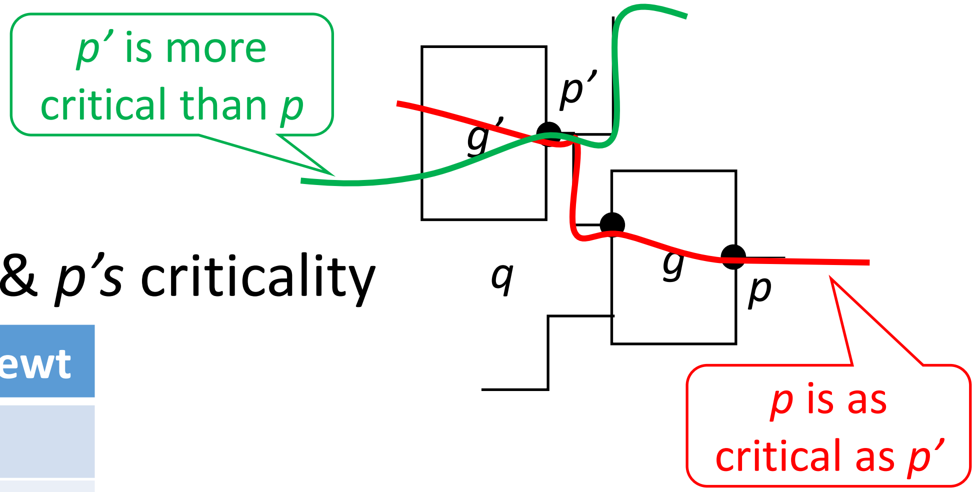
2. S. Held. "Gate sizing for large cell-based designs," in *DATE*'09.



2. S. Held. "Gate sizing for large cell-based designs," in *DATE*'09.



- Initialize slew targets as slews from STA
- Update slew targets
  - Globally critical: slack(*p*) < 0
  - Locally critical: whether *p* is on a critical path
  - Adjust the slew targets for *p* based on modes & *p’s* criticality



Gate position	Setup time slewt	Hold time slewt
Globally & locally critical	Decrease	Increase
Otherwise	Increase	Decrease

2. S. Held. “Gate sizing for large cell-based designs,” in *DATE*’09.

What values to update slew targets?

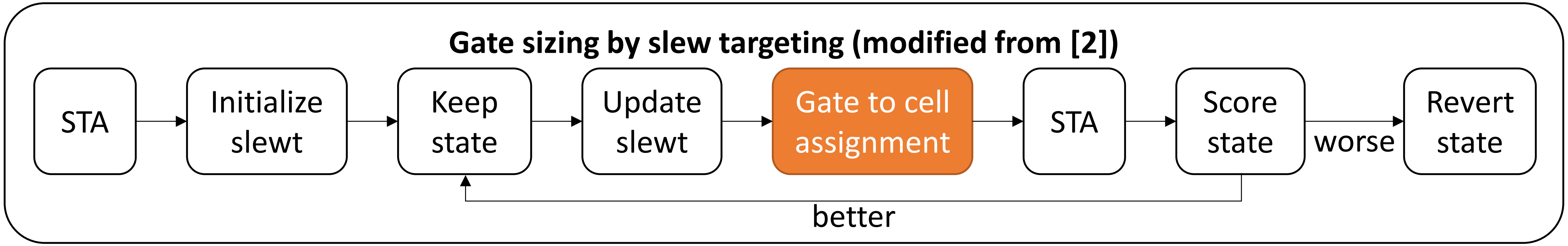
T\C	0.365616	1.895430	3.790860	7.581710	15.163400	30.326900	60.653700
1.23599	3.33809	5.59725	8.60523	14.8575	27.5164	52.8765	103.604
4.43724	3.33727	5.59699	8.60578	14.8576	27.5188	52.8775	103.599
15.6743	3.40246	5.62543	8.61689	14.8582	27.5170	52.8787	103.599
37.1331	4.36023	6.10464	8.84317	14.9465	27.5247	52.8726	103.605
70.5649	5.85455	7.27833	9.43026	15.0988	27.6409	52.9322	103.603
117.474	7.61897	9.14083	10.8314	15.5462	27.6912	53.0238	103.669
179.199	9.58764	11.3565	13.0249	16.7347	27.8716	53.0513	103.775

Output rising slew for BUF_X1, Nangate 45 nm, typical corner

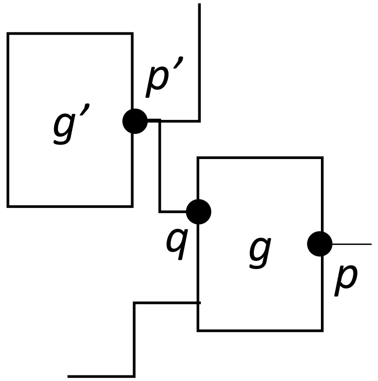
T\C	0.365616	3.786090	7.572190	15.144400	30.288800	60.577500	121.155000
1.23599	3.10917	5.67693	8.71288	14.9785	27.6350	52.9690	103.657
4.43724	3.10875	5.67786	8.71402	14.9788	27.6339	52.9719	103.660
15.6743	3.20354	5.70984	8.72471	14.9811	27.6310	52.9744	103.651
37.1331	4.20264	6.15463	8.94062	15.0761	27.6468	52.9670	103.666
70.5649	5.70174	7.27713	9.47332	15.2076	27.7634	53.0379	103.659
117.474	7.47026	9.13720	10.8172	15.6132	27.8134	53.1232	103.735
179.199	9.44195	11.3787	12.9969	16.7387	27.9813	53.1620	103.831

Output rising slew for BUF_X2, Nangate 45 nm, typical corner

- Slew possibilities
  - Values by table lookup into the slew table w/ current slew & different cap.
  - Upper bound (*ub*): cap. = max cap. of the pin
  - Lower bound (*lb*): cap. = 0
  - Values considered: *lb**(*ub*/*lb*)^(*n*/*k*)
    - In ParallelClosure, *k* = 20; *n* = 0, 1, 3, 5, 8, 11, 15, 20
- Update slew targets of pin *p* based on
  - Setup/hold time mode
  - *p*'s criticality & previous slew targets
- No max slew violation by construction



- Order of sizing
  - Want to fix fanout gates of *g* before sizing *g*
    - Output load matters more than input slew
  - Reverse topological order for gates
    - Cut cycles of gates at edges to register data inputs
- Slew estimation: see [2] for details

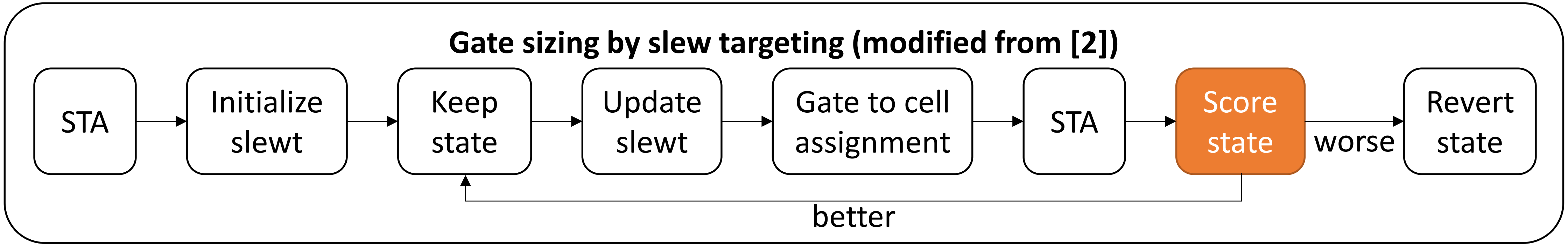


2. S. Held. "Gate sizing for large cell-based designs," in *DATE*'09.

How to select cells for gates?

Mode	For a given corner *cnr*	Across corners
Setup time	The smallest size that satisfies all slew targets	$size_s(g) = \max_{\forall cnr}\{size_{s,cnr}(g)\}$
Hold time	The largest size that satisfies all slew targets	$size_h(g) = \min_{\forall cnr}\{size_{h,cnr}(g)\}$

- If $\text{size}_s(g) \le \text{size}_h(g)$, assign *g* to the cell of size $\text{size}_s(g)$
  - Reduce area & leakage power
- If $\text{size}_s(g) > \text{size}_h(g)$, assign *g* to the cell of size $\text{size}_h(g)$
  - Honor hold time constraints while limiting the impact to setup time



- The new cell assignment (state) is better if
  - The worst negative slack improves for all corners and modes; or
  - The area is reduced w/o the following metrics significantly worsened in any corner and mode:
    - Worst negative slack
    - Average total negative slack over all path endpoints, e.g., register data inputs

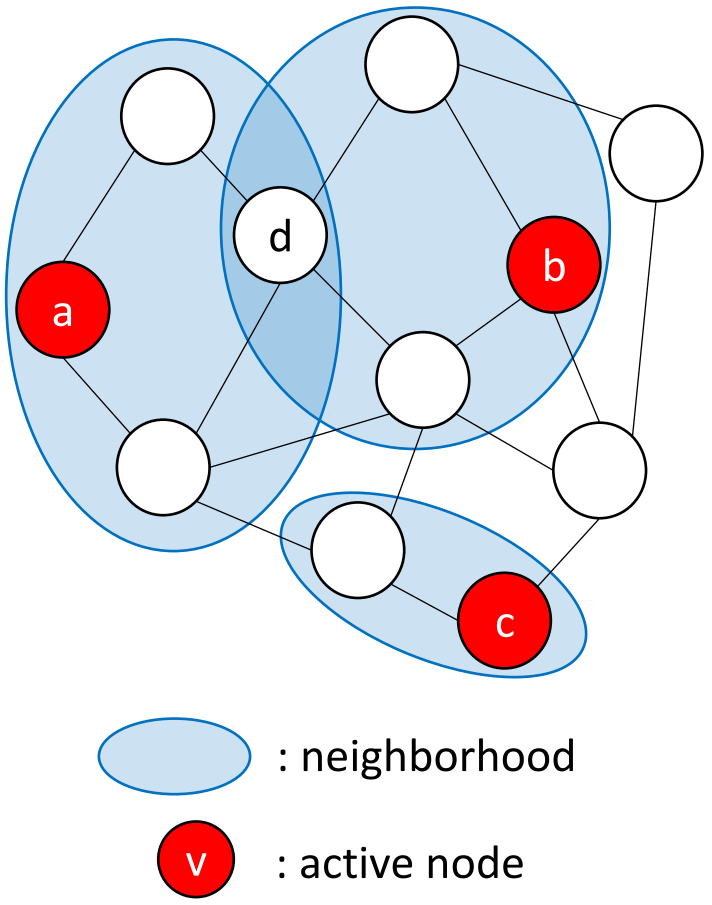
2. S. Held. “Gate sizing for large cell-based designs,” in *DATE*’09.

Outline

- Optimization flow – the algorithms
- Parallelization – boosting tool runtime
- Limitation
- Conclusions

Parallelization w/ operator formulation [3]

- Active elements
  - Nodes/edges/subgraphs where computation is needed
- Operator
  - Computation at active elements
  - Neighborhood: set of nodes/edges read/written by the update
  - Morph operators may change graph topology
  - Label-computation operators only update node/edge labels
- Schedules
  - The ordering to apply operators on active elements
  - May be constrained for correctness
  - Some ordering may perform better than the others
- Parallelism
  - Disjoint updates
  - Read-only operators



3. K. Pingali et al. "The TAO of parallelism in algorithms," in PLDI'11.

Shared-memory Galois: A C++ library for operator formulation of algorithms [4]

Features of Galois

- Parallel data structures
  - Graphs, bags, etc.
- Parallel loops over active elements
  - for_each, do_all, etc.
- Support for
  - Load balancing
  - Scheduling
  - Dynamic work
  - Transactional execution

Successes in EDA

- FPGA routing
  [Moctar & Brisk, *DAC* 2014]
- AIG rewriting
  [Possani et al., *ICCAD* 2018]
- Timing closure
  [Lu et al., TAU 2019 contest]

4. D. Nguyen, A. Lenharth, K. Pingali. "A lightweight infrastructure for graph analytics", in *SOSP*'13.

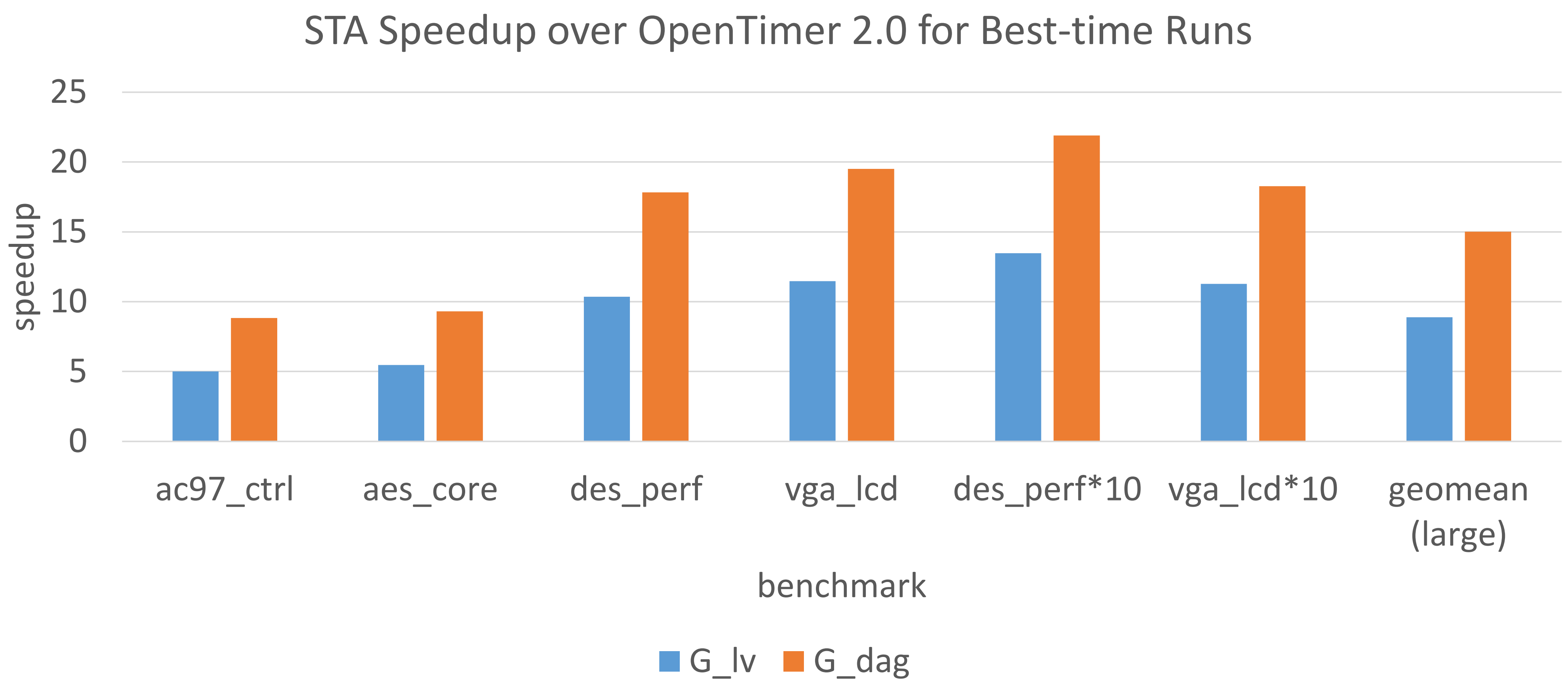
How to write a timer in Galois

Core functionality	LOC in total	LOC for parallelization
STA	391	35 (8.91%)
Gate sizing	639	97 (15.18%)
Buffering	439	35 (7.99%)

```
#include “TimingGraph.h”
// other header includes

using GNode = TimingGraph::GraphNode;
using GNodeBag = galois::InsertBag<GNode>;

void propagateForward(TimingGraph& g) {
  GNodeBag fFront;
  initForward(g, fFront);
  computeForward(g, fFront);
}

// other codes for propagateBackward
//   & reportCriticalPath

int main(int argc, char** argv) {
  galois::SharedMemSys G;

  // instantiate a timing graph
  TimingGraph g;
  // construct g using cell libraries
  //   & Verilog netlist
  // initialize g using SDC commands

  propagateForward(g);
  propagateBackward(g);
  reportCriticalPath(g);
  return 0;
}
```

```
void initForward(TimingGraph& g, GNodeBag& bag)
{
  bag.clear();
  galois::do_all(
    galois::iterate(g),
    [&] (GNode n) {
      auto inDeg = inDegree(n);
      g.getData(n).dep = inDeg;
      if (!inDeg) {
        bag.push_back(n);
      }
    }
    , galois::loopname(“InitForward")
    , galois::steal()
  );
}
```

```
void computeForward(TimingGraph& g, GNodeBag& bag) {
  galois::for_each(
    galois::iterate(bag),
    [&] (GNode n, auto& ctx) {
      computeForwardOperator(n, ctx.getPerIterAlloc());

      // schedule an outgoing neighbor when required
      for (auto e: g.edges(n, unprotected)) {
        auto succ = g.getEdgeDst(e);
        auto& succData = g.getData(succ);
        if (!__sync_sub_and_fetch(&(succData.dep), 1)) {
          ctx.push(succ);
        }
      }
    }
    , galois::loopname("ComputeForward")
    , galois::per_iter_alloc()
    , galois::no_conflicts()
  );
}
```



Circuit	# Gates	# Nets	# Pins	Sequential Runtime			Best Runtime (and # Threads Used)			Speedup over *OT*	
				OT	G_{lv}	G_{dag}	*OT*	G_{lv}	G_{dag}	G_{lv}	G_{dag}
ac97_ctrl	14,131	14,407	40,238	390.0	114.0	101.3	312.0 (21)	62.3 (7)	35.3 (7)	5.01	8.83
aes_core	22,938	23,199	66,221	623.3	226.3	196.7	493.3 (7)	90.3 (7)	53.0 (7)	5.46	9.31
des_perf	105,371	106,532	295,808	3,453.0	1,173.3	956.3	2,762.7 (14)	266.7 (14)	155.0 (14)	10.36	17.82
vga_lcd	139,529	139,631	380,730	4,700.3	1,495.7	1,232.3	3,660.7 (28)	319.3 (14)	187.7 (14)	11.46	19.51
des_perf*10	1,053,710	1,065,311	2,958,071	34,853.0	12,765.3	10,441.7	29,923.3 (14)	2,222.0 (14)	1,366.7 (14)	13.47	21.90
vga_lcd*10	1,395,290	1,396,301	3,807,291	49,284.3	16,063.7	13,084.0	31,212.7 (35)	2,768.7 (14)	1,708.7 (14)	11.27	18.27

Outline

- Optimization flow – the algorithms
- Parallelization – boosting tool runtime
- Limitation
- Conclusions

Limitation

5. E. G. Friedman, “Clock distribution networks in synchronous digital integrated circuits,” in *Proc. of the IEEE*, *89*(5): pp. 665–692, 2001.
6. C. J. Alpert et al. “Buffered steiner trees for difficult instances,” in *IEEE/ACM TCAD, 21*(1): pp. 3–14, 2002.
7. L. P. P. P. van Ginneken. “Buffer placement in distributed rc-tree networks for minimal elmore delay,” in *ISCS*’90.

Quality of results

- Lots of buffers are inserted when there is a large number of paths w/ hold-time violations
  - Clock network synthesis [5] may help
- Not considering net topology and optimal buffer insertion for a net
  - Topology: C-tree algorithm [6]
  - Optimal buffer insertion: van Ginneken’s algorithm [7]
- Need more parameter tuning
  - E.g., convergence criteria of sizing

Performance of ParallelClosure

- Buffer insertion is purely sequential
  - Consistency of name-object mappings
  - The algorithm for fixing hold-time violations has no parallelism

Outline

- Optimization flow – the algorithms
- Parallelization – boosting tool runtime
- Limitation
- Conclusions

Conclusions

1. N. V. Shenoy, R. K. Brayton, A. L. Sangiovanni-Vincentelli. "Minimum padding to satisfy short path constraints," in *ICCAD*'93.
2. S. Held. "Gate sizing for large cell-based designs," in *DATE*'09.
3. K. Pingali et al. "The TAO of parallelism in algorithms," in *PLDI*'11.
4. D. Nguyen, A. Lenharth, K. Pingali. "A lightweight infrastructure for graph analytics," in *SOSP*'13.

- ParallelClosure is effective for designs w/ a small # hold-time violations
  - Buffer insertion for fixing hold time violations [1]
  - Gate sizing by slew targeting [2] for minimizing area, leakage power & clock period
  - All algorithms are generalized for multi-corner, multi-mode (MCMM) optimizations
- ParallelClosure is efficient through parallelizing STA & gate sizing
  - Parallelism analyses using the operator formulation [3]
  - Parallel implementation using the shared-memory Galois framework [4]

Thanks!

Questions? Comments?