SPRoute 2.0:
A detailed-routability-driven deterministic parallel global router with soft capacity

Jiayuan He[1], Udit Agarwal[2], Yihang Yang[3], Rajit Manohar[3], Keshav Pingali[1]

[1]Computer Science, UT Austin
[2]Katana Graph, Inc
[3]Electrical Engineering, Yale University

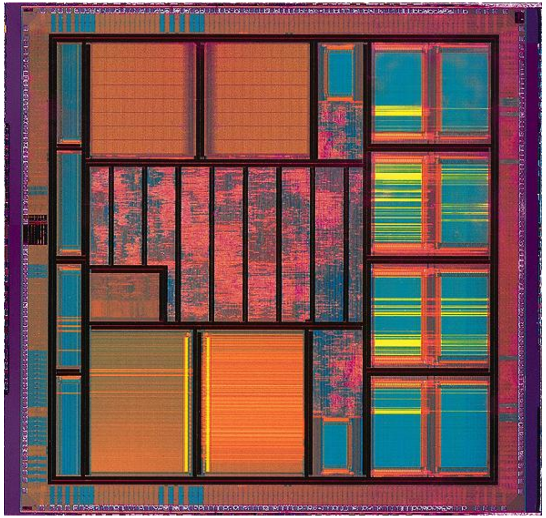
Overview

- Introduction
- Algorithms
  - Detailed-routability
  - Deterministic parallelization
- Experimental results
- Conclusion

Introduction

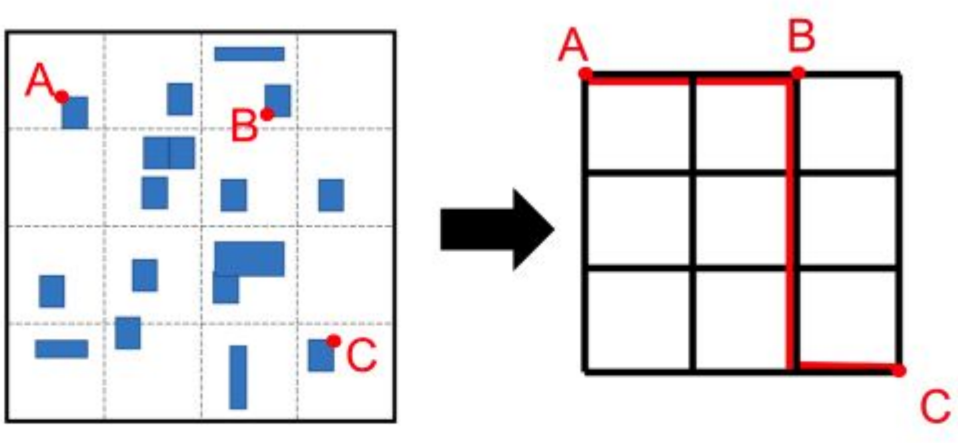
- VLSI design
  - Complex design rules in advanced technology nodes
  - Increasing size of chips

- Global routing
  - Detailed routability
  - Deterministic and fast parallelization



Introduction: global routing

- Partition routing space to GCells
- Grid graph
  - Edge capacity: # available routing tracks



Previous work

- Detailed routability
  - FastRoute 4.0 [Xu+, ASPDAC’09], NCTU-GR 2.0[Liu+, TCAD’13], NTHU-Route[Chang+, ICCAD’08], NTUgr[Chen+, ASPDAC’09]: route on the grid graph
  - VFGR [Cai+, ASPDAC’14]: congestion model for layout components and capacity on node
  - CUGR[Liu+, DAC’20] : 3D pattern routing, probabilistic resource model for detailed routability
  - Limitation: DRCs
- Parallelization on maze routing
  - NCTU-GR 2.0 [Liu+, TCAD’13]: collision-aware rip-up and reroute
  - SPRoute [He+, ICCAD’19]: two-phase maze routing
  - Limitation: non-determinism

Contributions

- Detailed routability
  - Soft capacity: reserve routing space based on congestion
  - Congestion is estimated by pin density and net density (RUDY)
  - Reduce 43% shorts and 14% DRCs
- Deterministic parallelization
  - Bulk synchronous maze routing
  - Scheduler to reduce load imbalance and livelock
  - 7.4X faster than state-of-the-art

Overview

- Introduction
- Algorithms
  - Detailed-routability-driven
  - Deterministic parallelization
- Experimental results
- Conclusion

Algorithms

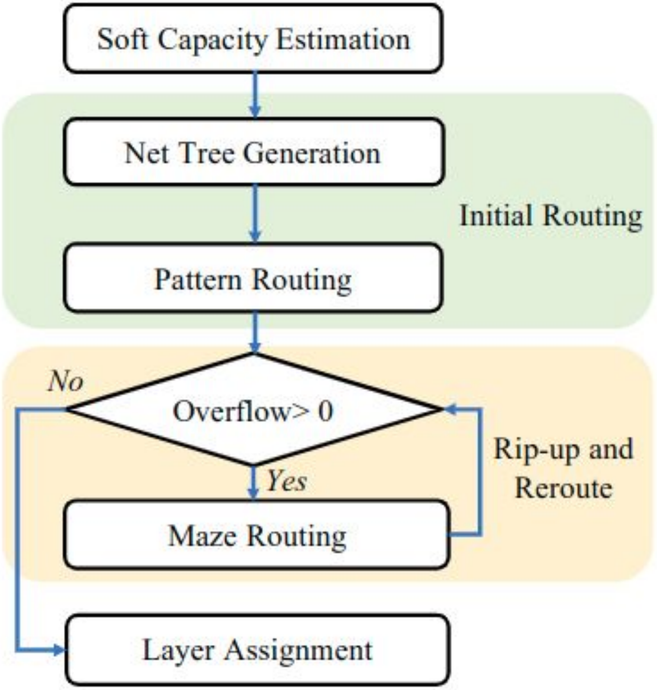


Fig 1. Overall flow of SPRoute 2.0

Overview

- Introduction
- Algorithms
  - Detailed-routability
  - Deterministic parallelization
- Experimental results
- Conclusion

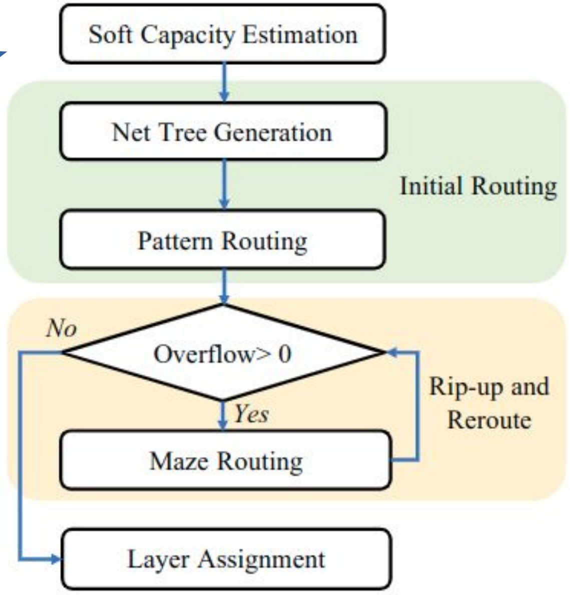
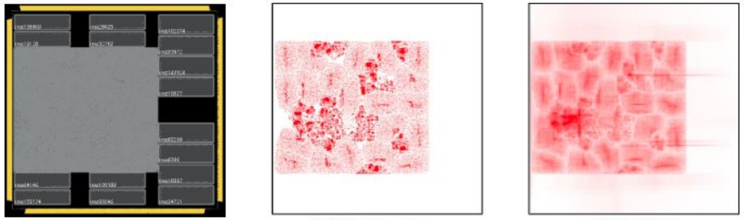
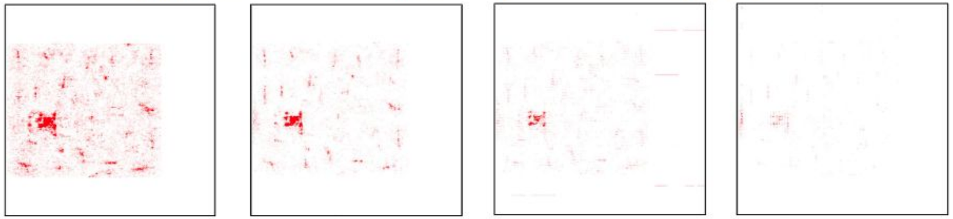


Fig 1. Overall flow of SPRoute 2.0

Soft capacity example



(a) ispd19_test7 (b) Pin Density (c) RUDY



(d) Metal2 DRCs (e) Metal3 DRCs (f) Metal4 DRCs (g) Metal5 DRCs

Fig 2. Heat map of pin density, RUDY and DRCs

- RUDY [Spindler+, DATE’07]:
  - Rectangular Uniform wire DensitY
  - RUDY = HPWL/Bounding box area
- Take-aways:
  - DRCs are related to pin density and wire density (RUDY)
  - Low metal layers are more affected by congestion

Soft capacity estimation

- Congestion:

$$cong\left(x,y\right) = pin\_density\left(x,y\right) + w * RUDY\left(x,y\right)$$

- Soft capacity:

$$soft\_cap\left(x,y\right) = ratio\left(\overline{cong}\left(x,y\right)\right) * hard\_cap\left(x,y\right)$$

- Ratio:

$$ratio\left(cong\right) = min + \frac{max-min}{1+exp((cong-cong_{mid})*k)}$$

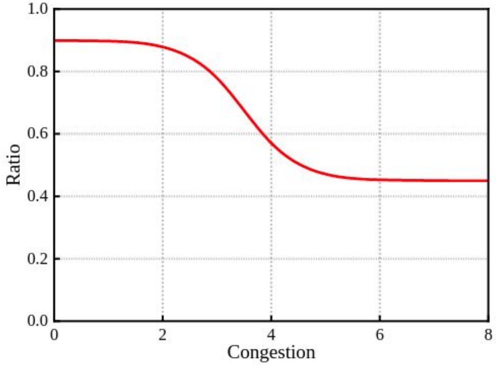


Fig 3. Ratio Function

Edge cost function

Maze routing: shortest path problem on the grid graph

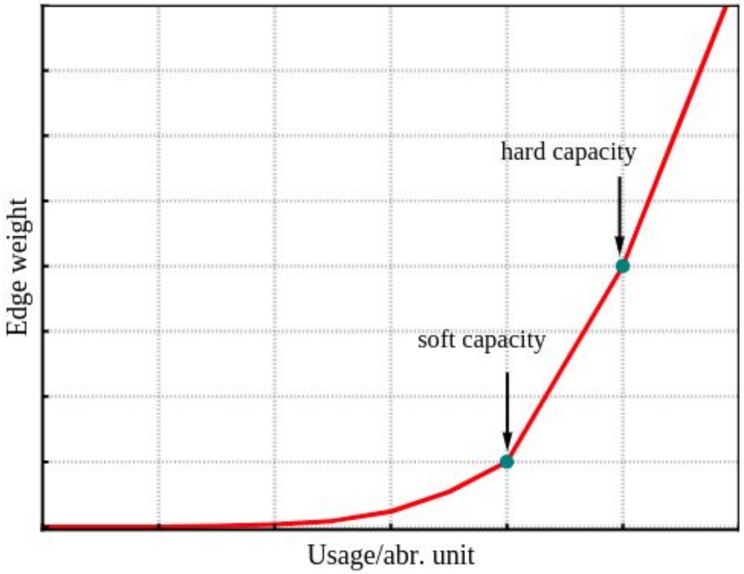


Fig 4. Three-stage Cost Function

Overview

- Introduction
- Algorithms
  - Detailed-routability-driven
  - Deterministic parallelization
- Experimental results
- Conclusion

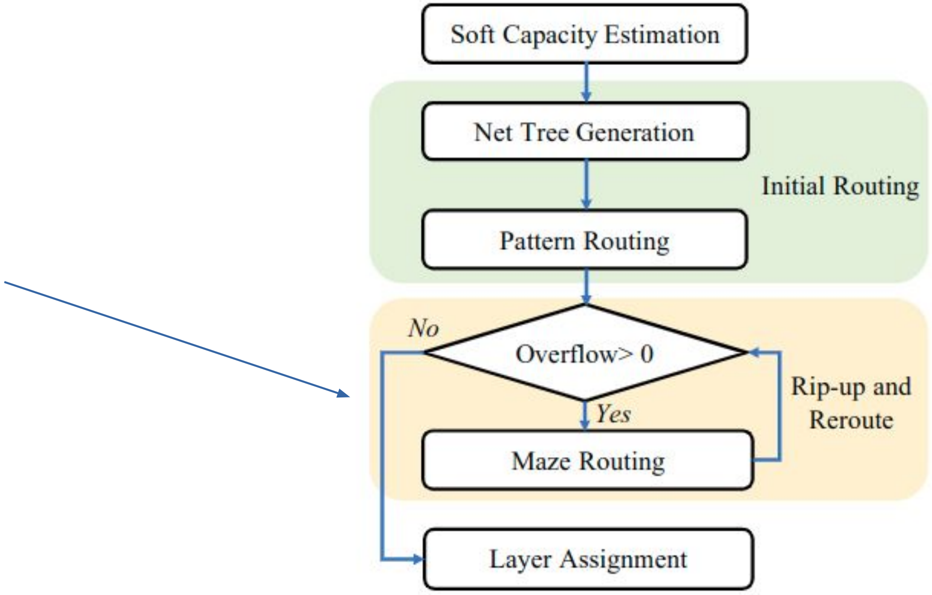


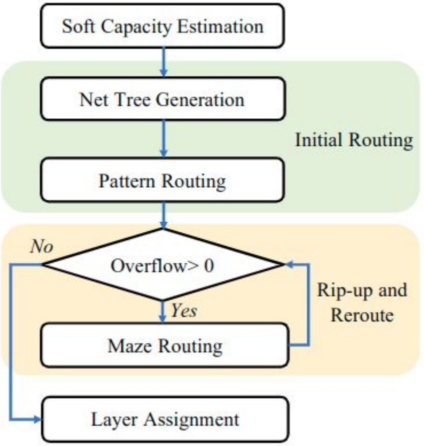
Fig 1. Overall flow of SPRoute 2.0

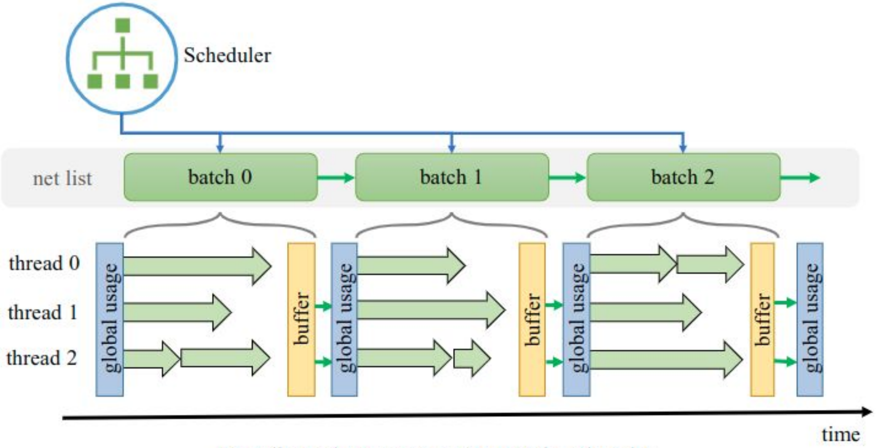
Deterministic parallel maze routing

- Routing regions of nets are highly overlapped
  - Speedup of parallelization is limited
- Non-deterministic parallel maze routing
  - NCTU-gr 2.0[Liu+, TCAD’13], SPRoute [He+, ICCAD’19]
  - Threads route nets through the same region concurrently

- Bulk synchronous deterministic maze routing
  - Does not require concurrent nets to be disjoint

Bulk synchronous maze routing

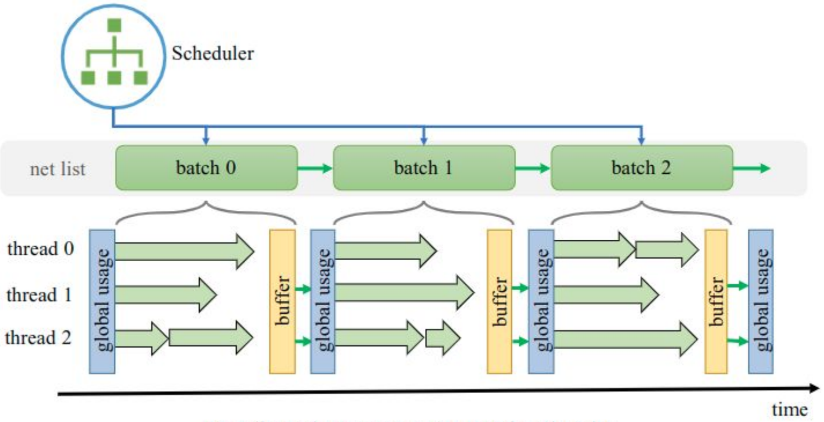




(a) Bulk synchronous execution and the scheduler

- Scheduler partitions netlist into batches
- All threads route nets from the same batch
  - Each thread acquires a net
  - Routes based on the global usage after the previous batch
  - Writes new usages into a batch-local buffer
- Buffer usages are updated to global usage after the completion of a batch

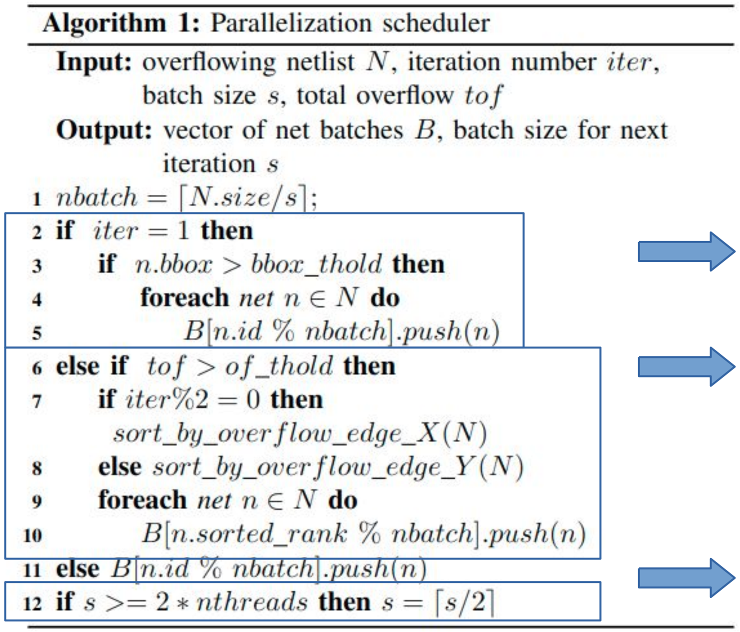
Performance Issues



(a) Bulk synchronous execution and the scheduler

- Load imbalance
  - Large and small nets in the same batch
  - Significant in the 1st iteration of maze routing
- Livelock
  - Nets are ripped up and rerouted repeatedly due to stale values
  - Factors: degree of overlap, same scheduling, batch size

Scheduler



```
Algorithm 1: Parallelization scheduler
Input: overflowing netlist $N$, iteration number $iter$,
       batch size $s$, total overflow $tof$
Output: vector of net batches $B$, batch size for next
        iteration $s$
$nbatch = \lceil N.size/s \rceil$;
if $iter = 1$ then
    if $n.bbox > bbox\_thold$ then
        foreach net $n \in N$ do
            $B[n.id \,\%\, nbatch].push(n)$
else if $tof > of\_thold$ then
    if $iter\%2 = 0$ then
     $sort\_by\_overflow\_edge\_X(N)$
    else $sort\_by\_overflow\_edge\_Y(N)$
    foreach net $n \in N$ do
        $B[n.sorted\_rank \,\%\, nbatch].push(n)$
else $B[n.id \,\%\, nbatch].push(n)$
if $s >= 2 * nthreads$ then $s = \lceil s/2 \rceil$
```

1. Filter out small nets in the 1st iteration
(load imbalance)

2. Sort the net by X or Y coordinate alternatively and schedule close nets to different batches
(livelock)

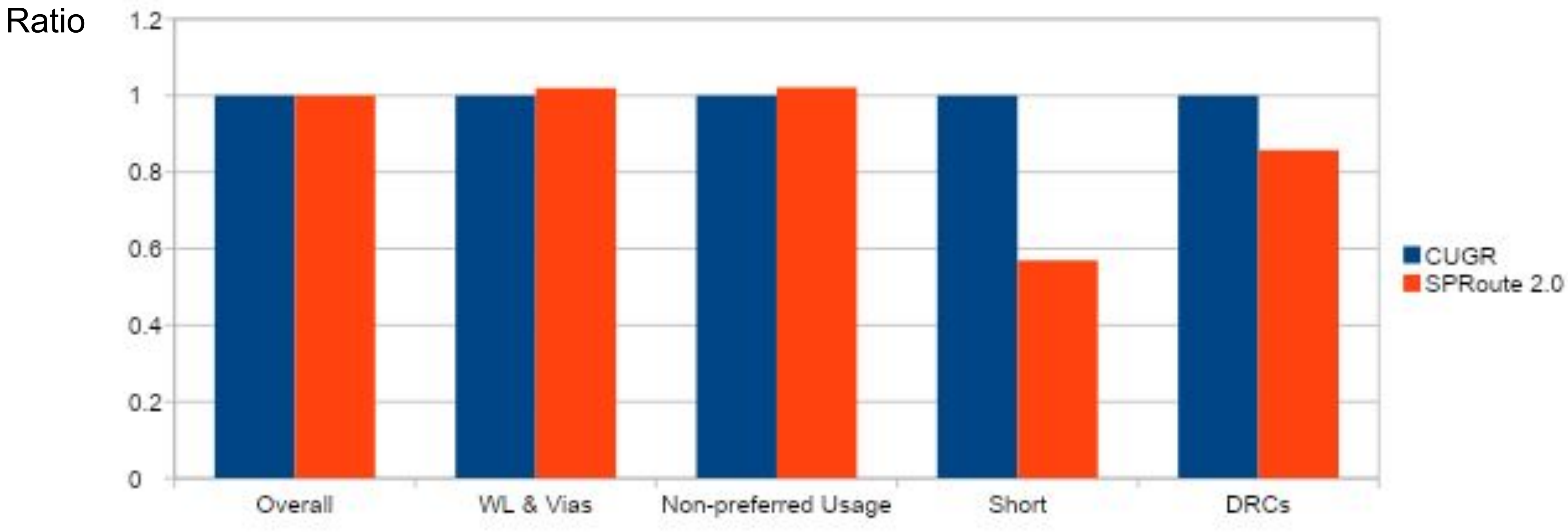
3. Reduce batch size, critical to the overflow convergence
(livelock)

Experiment Setup

- SPRoute 2.0 implemented in C++
- Optimized for 8 thread
- Benchmarks: ICCAD’19 contest
- Metrics:
  - Quality: Weighted score including wirelength, vias, non-preferred usages, DRCs and shorts
  - Runtime: 1 to 8 threads
- Baseline: CUGR [Liu+, DAC’20]

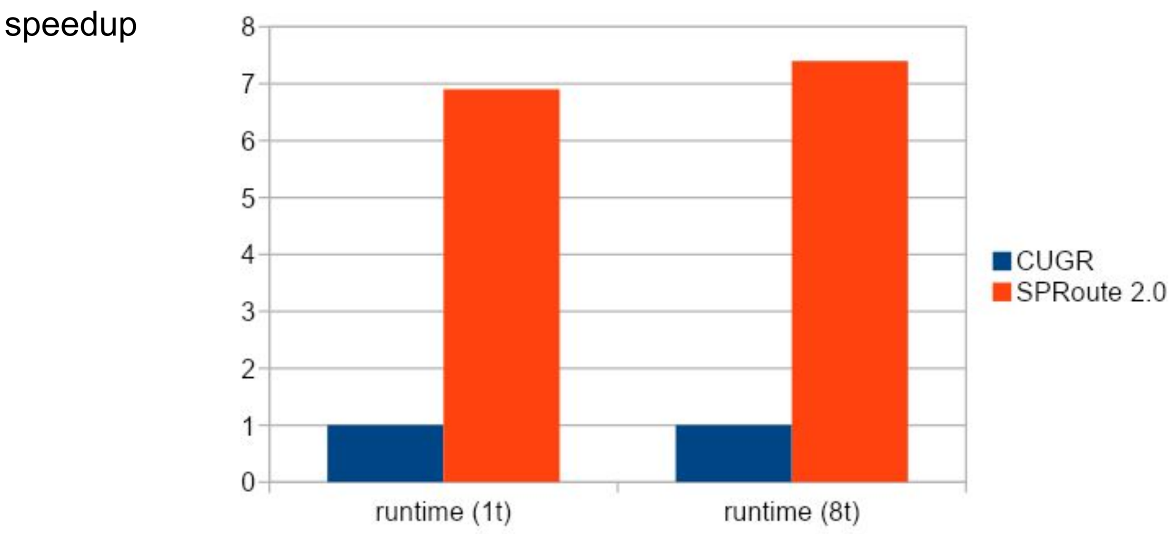
Quality

- Overall: 100.1%
- Wirelength & Vias: 102% Non-preferred Usage: 102.1%
- Shorts: 57% DRCs: 85%



Runtime

- 6.9X faster with 1 thread
- 7.4X faster with 8 threads



Scalability

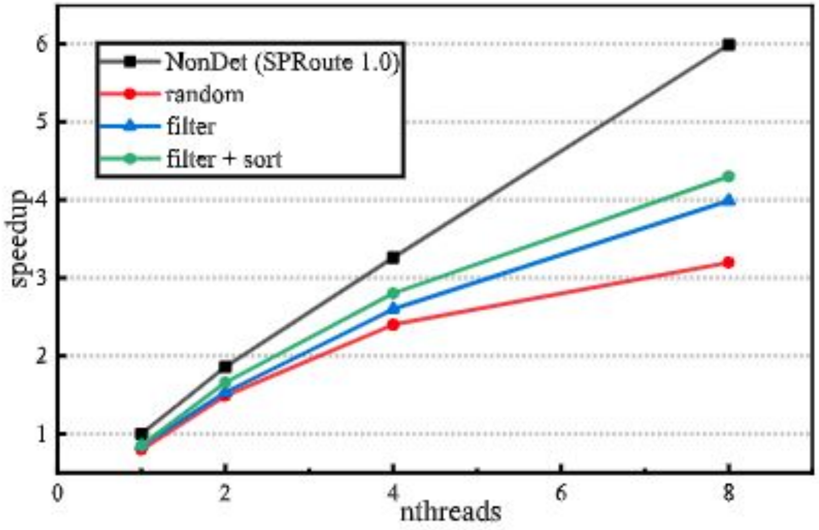


Fig. Speedup of maze routing

Scheduler algorithm	speedup
Random partition + batch reduction:	3.2X
+ Filter:	4.0X
+ Filter + sort:	4.3X

Conclusions

- Detailed Routability
  - Soft capacity: reserve routing space based on congestion
  - Congestion is estimated by pin density and net density (RUDY)
  - Reduce 43% shorts and 14% DRCs
- Deterministic Parallelization
  - Bulk synchronous maze routing
  - Scheduler to reduce load imbalance and livelock
  - 7.4X faster than state-of-the-art

Thank you!