

Open source placement and routing tools for FPGAs and their extensions to multi-die FPGAs

Prof. Dirk Stroobandt

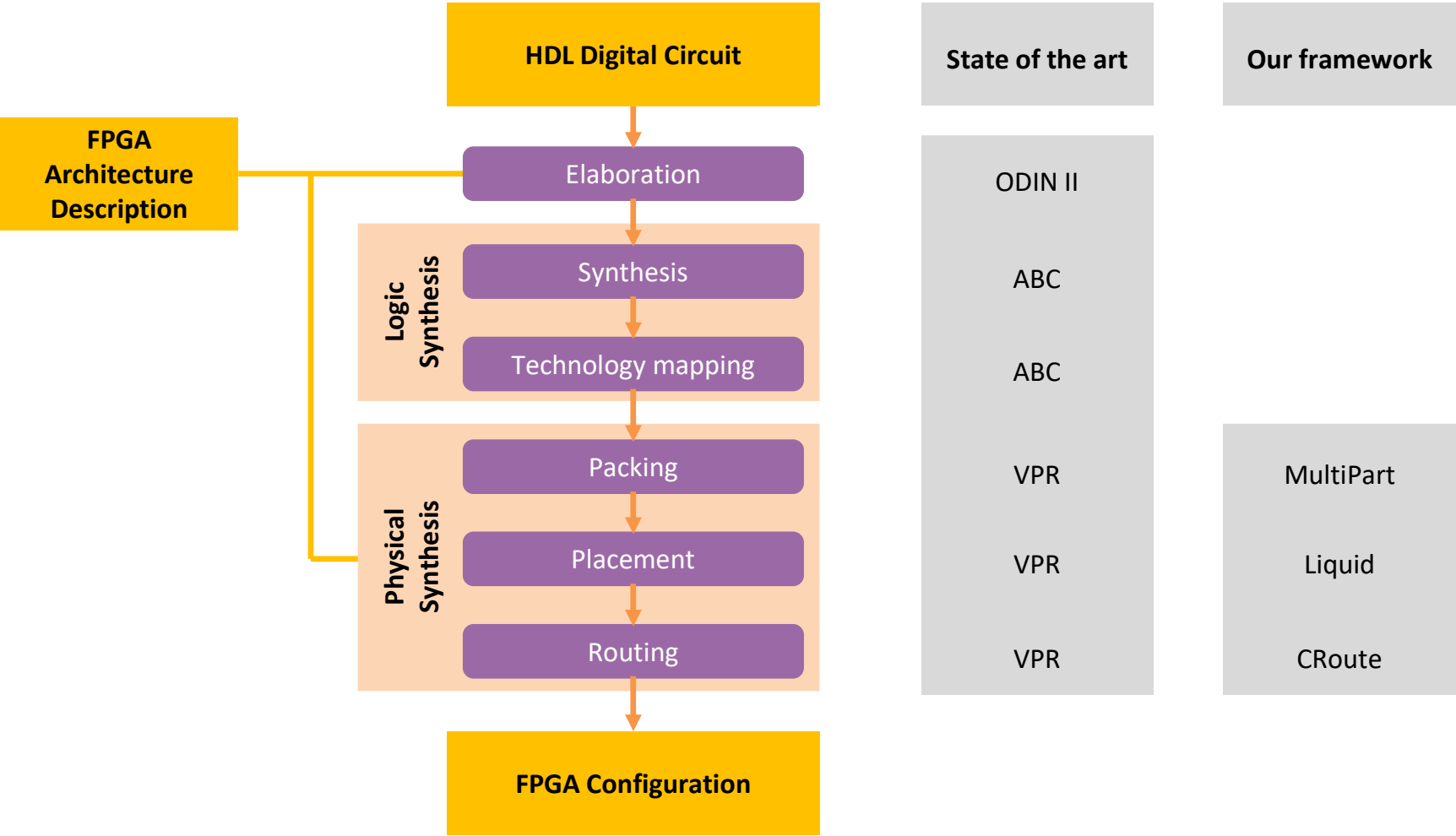
Ghent University, Belgium

Presentation overview

- Connection-based physical design
 - Our FPGA CAD framework
 - Liquid placement tool
 - Croute and RWRoute
- Multi-die placement and routing
 - LiquidMD
 - CRouteMD (in progress)

FPGA CAD FRAMEWORK

FPGA CAD FRAMEWORK



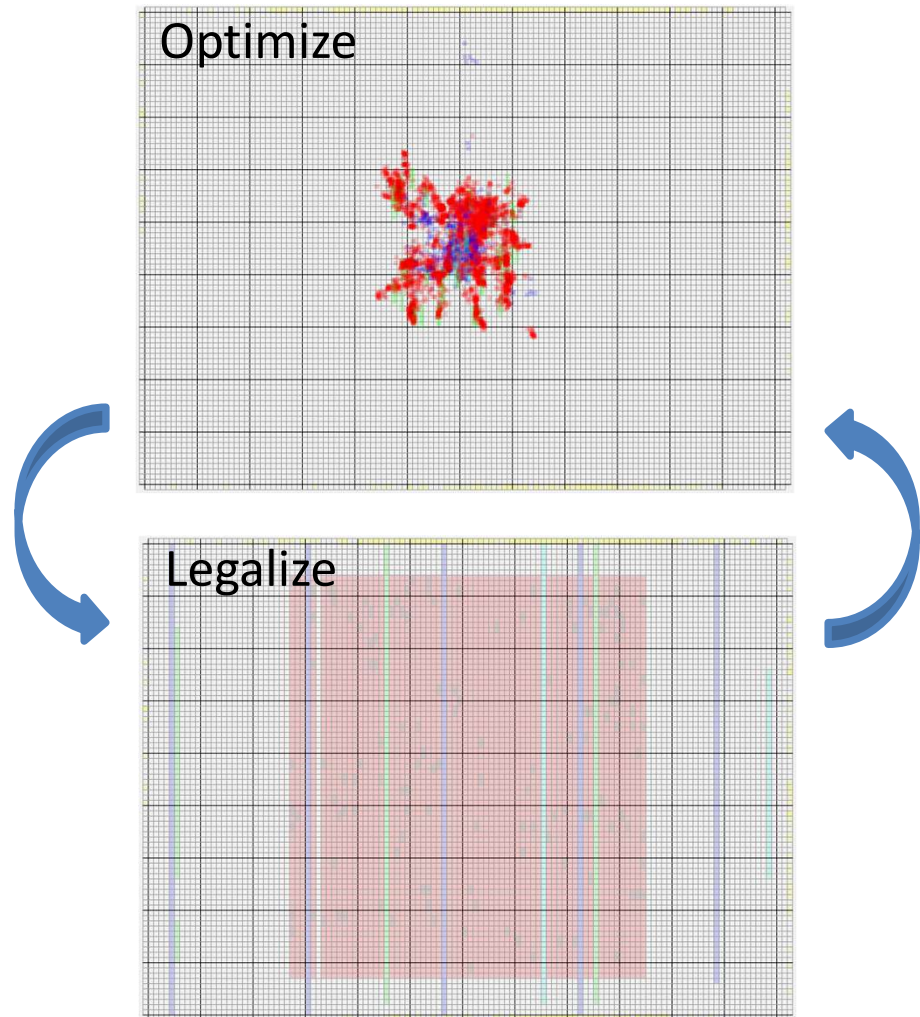
FPGA CAD FRAMEWORK

		State of the art	Our framework	1/RT	CPD	WL
Physical Synthesis	Packing	VPR	MultiPart	8.4x	-8%	-32%
	Placement	VPR	Liquid	28x	-1%	-2%
	Routing	VPR	CRoute	3.4x	-6%	-11%

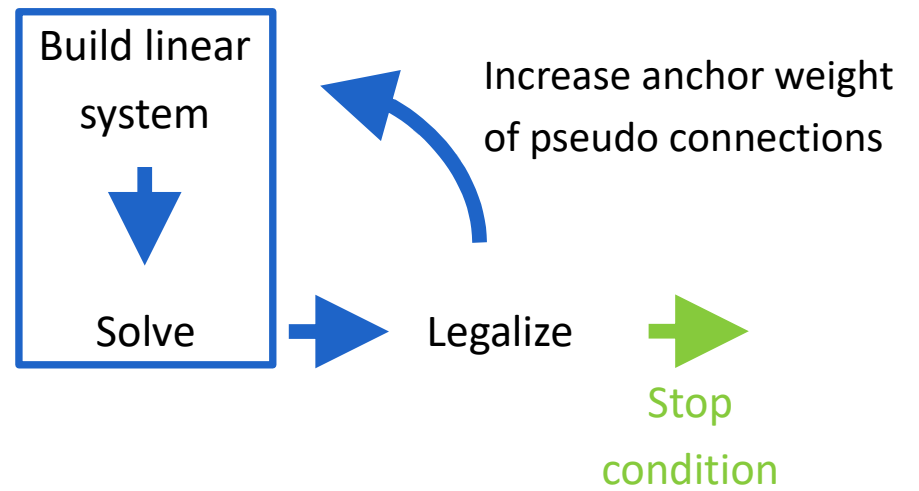
<https://GitHub.UGent.be/UGent-HES/FPGA-CAD-Framework>

Liquid:
High-quality Scalable
Placement for Large
Heterogeneous FPGAs

Placement problem

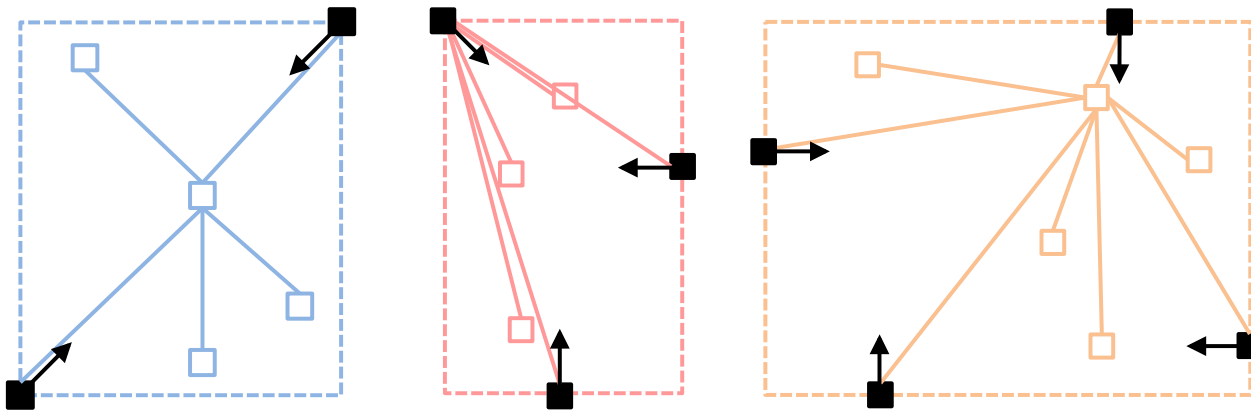
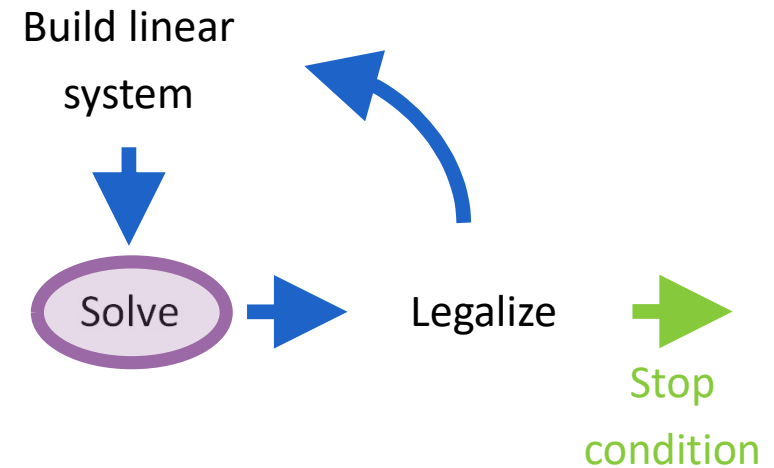


Analytical placement



Liquid placement technique

- Each net is spring between extreme blocks
- Long springs pull harder
- Extra spring for highly critical source-sink connections



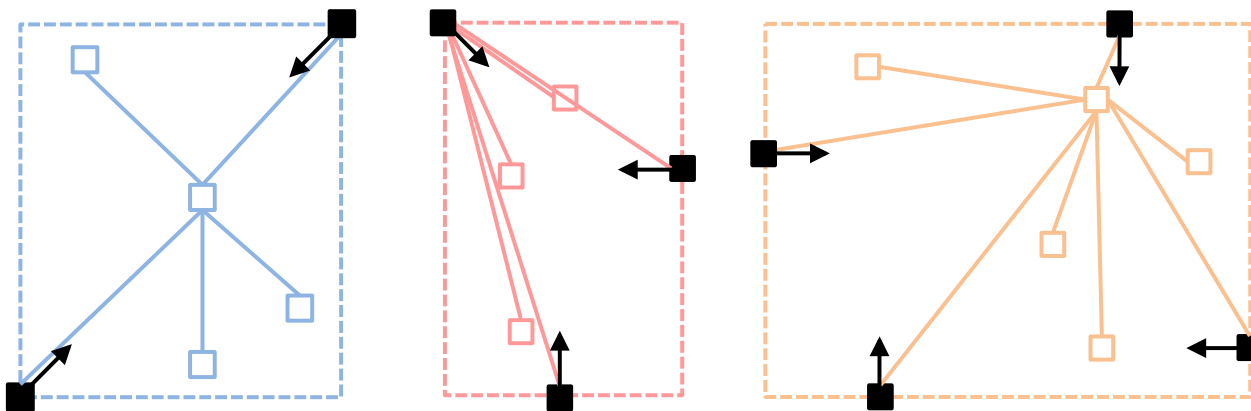
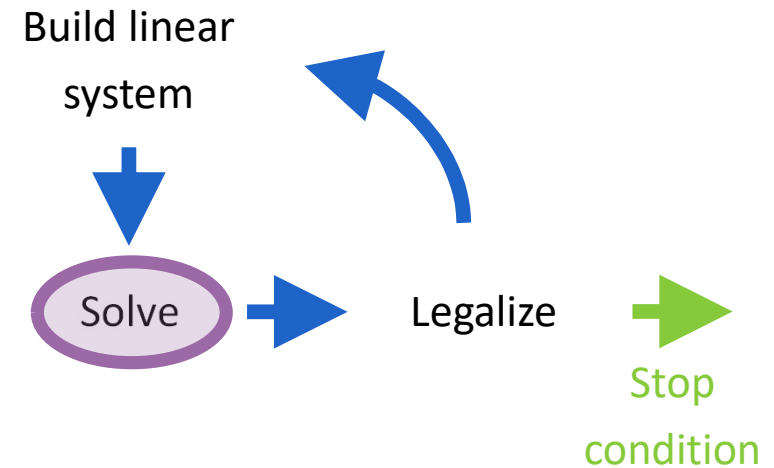
E. Vansteenkiste, S. Lenders, and D. Stroobandt, "Liquid: Fast placement prototyping through steepest gradient descent movement," in Field Programmable Logic and Applications (FPL), 2016 26th International Conference on. IEEE, 2016, pp. 1–4.

Liquid placement technique

Is it necessary to exactly solve the linear system?

➡ Legalization partly destroys the solution

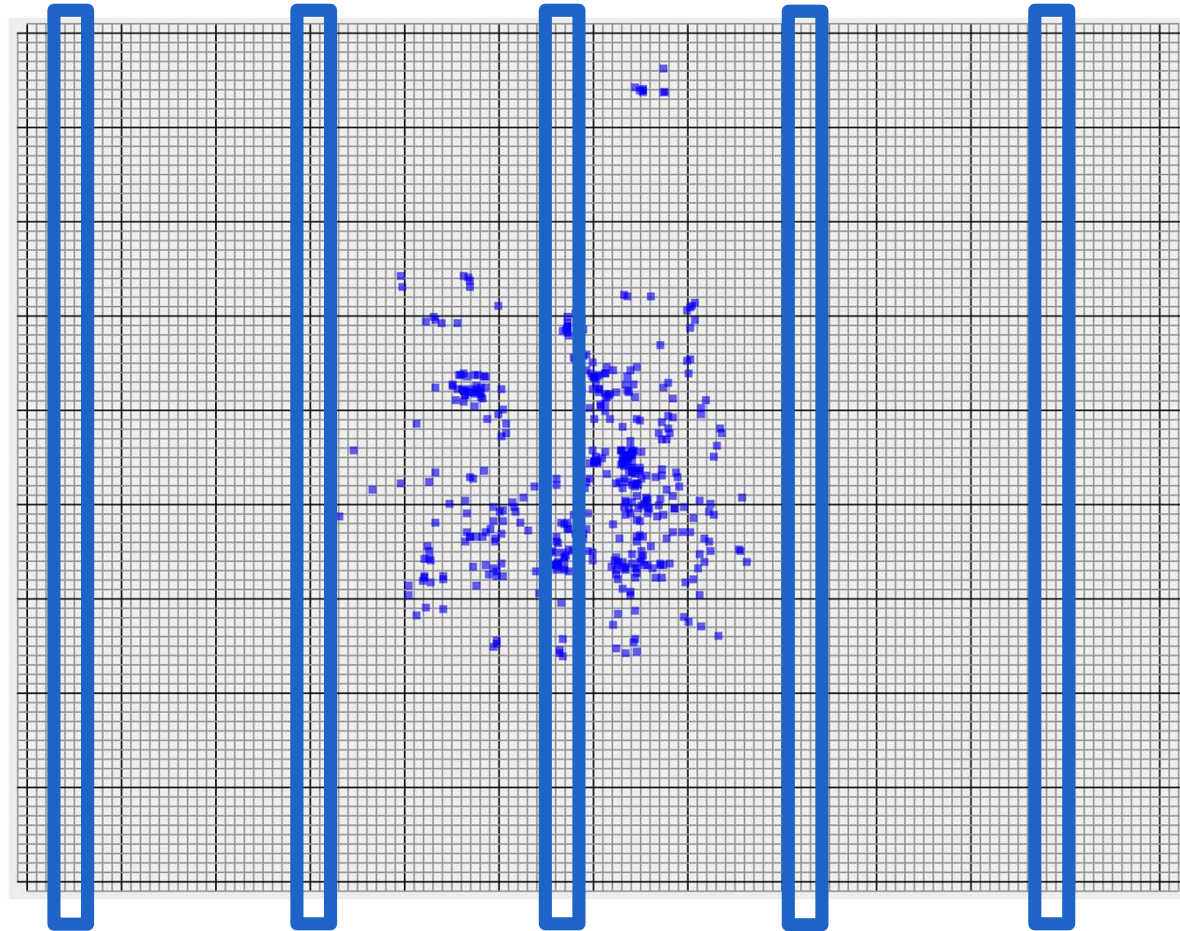
Optimize system by moving each block several times in the direction that reduces the placement cost the most



E. Vansteenkiste, S. Lenders, and D. Stroobandt, "Liquid: Fast placement prototyping through steepest gradient descent movement," in Field Programmable Logic and Applications (FPL), 2016 26th International Conference on. IEEE, 2016, pp. 1–4.

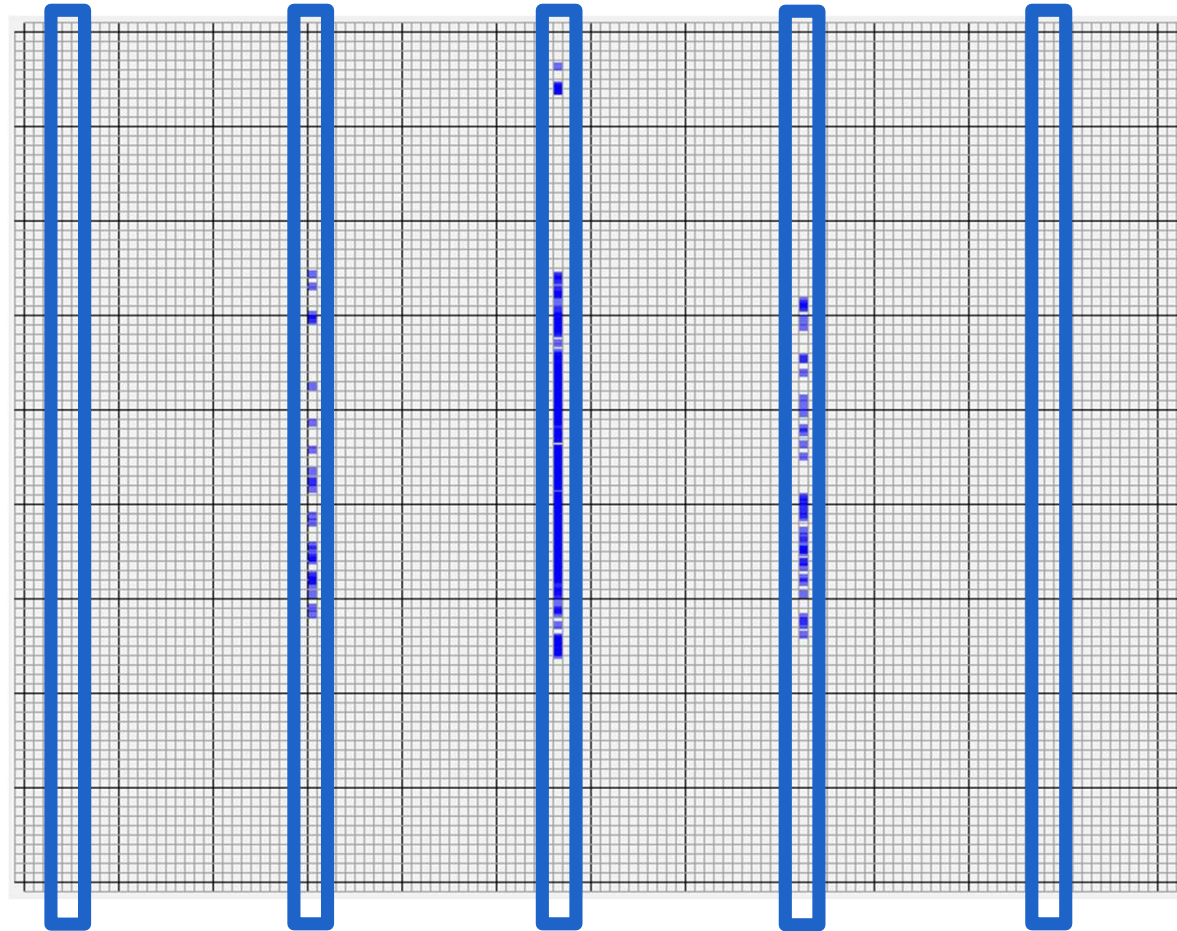
Hard Block Legalization

Optimized placement



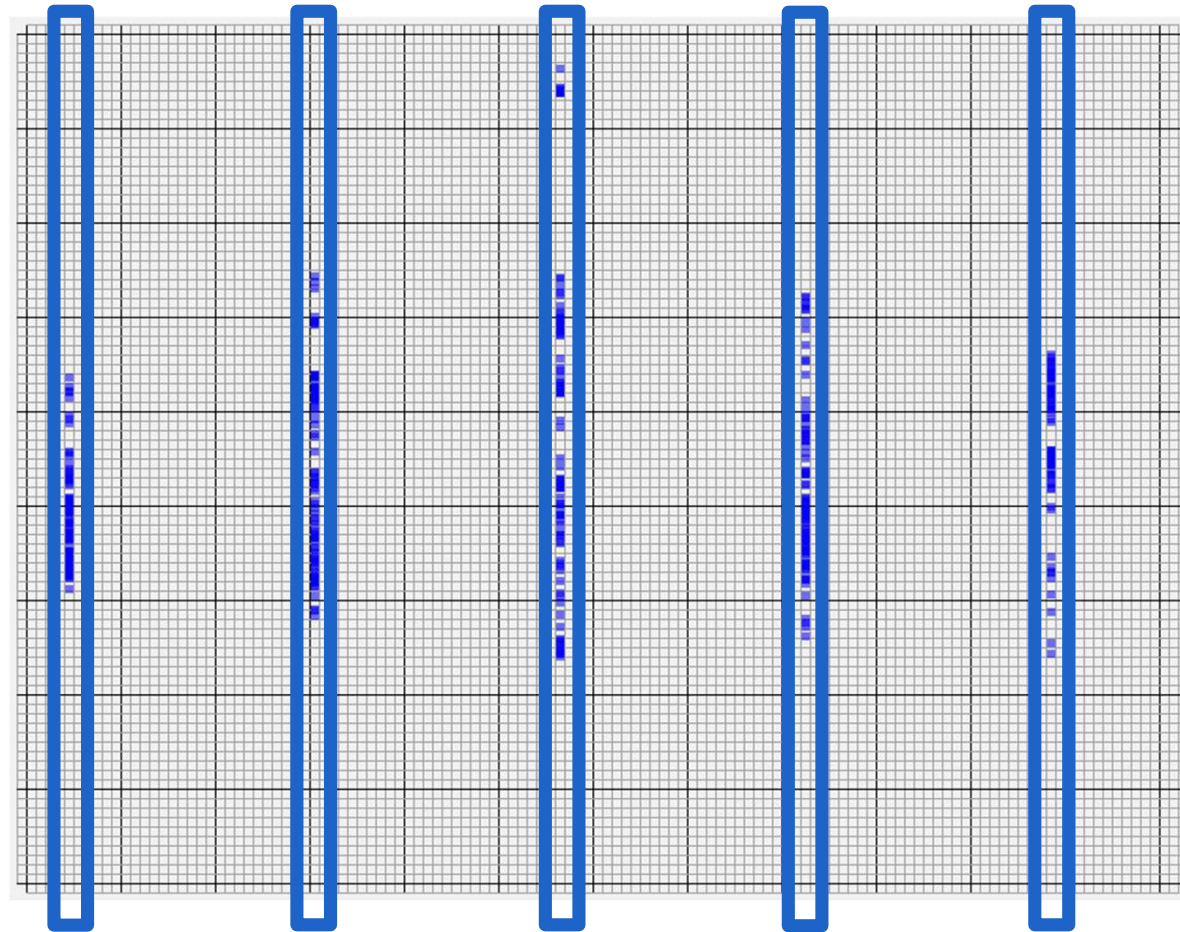
Hard Block Legalization

1. Assign to nearest hard block column



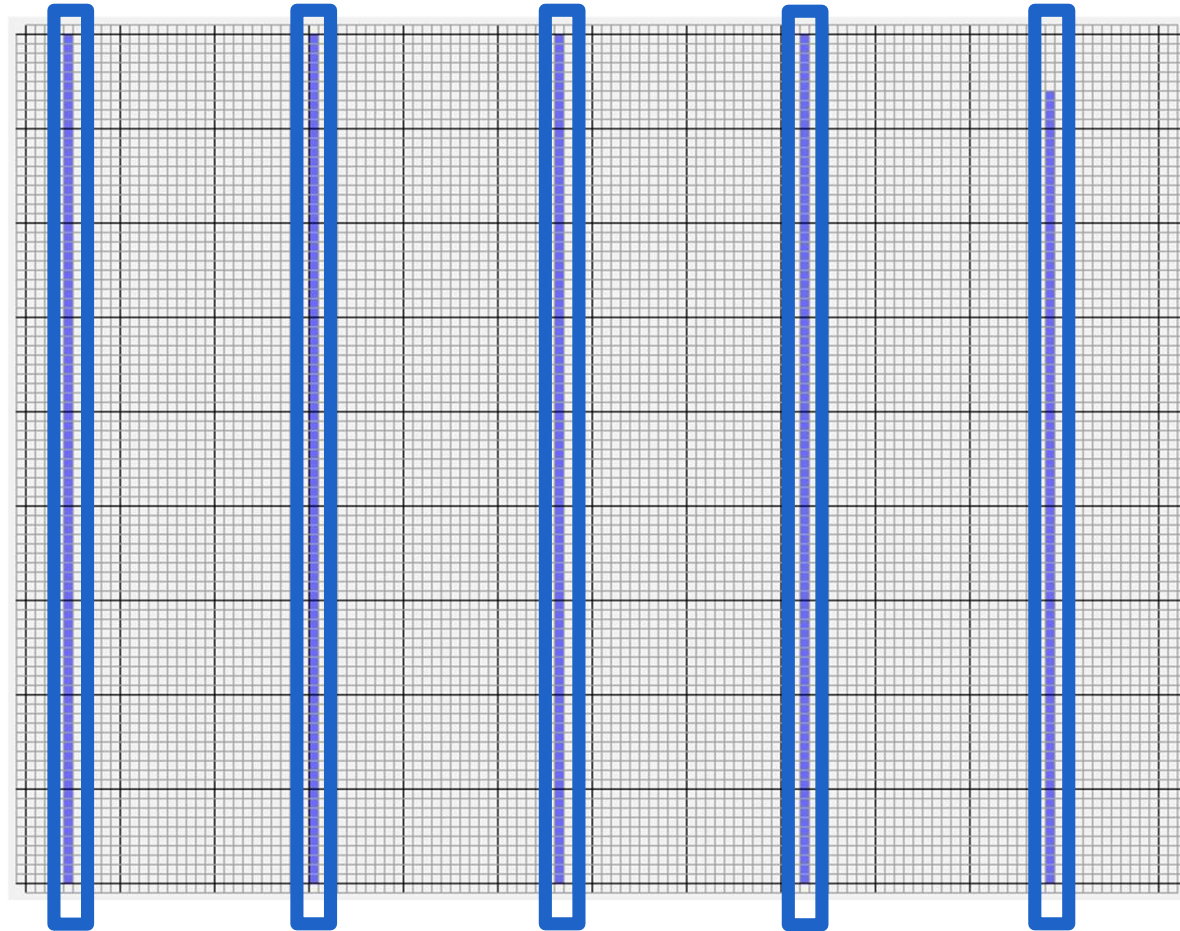
Hard Block Legalization

2. Column swap → satisfy capacity constraints



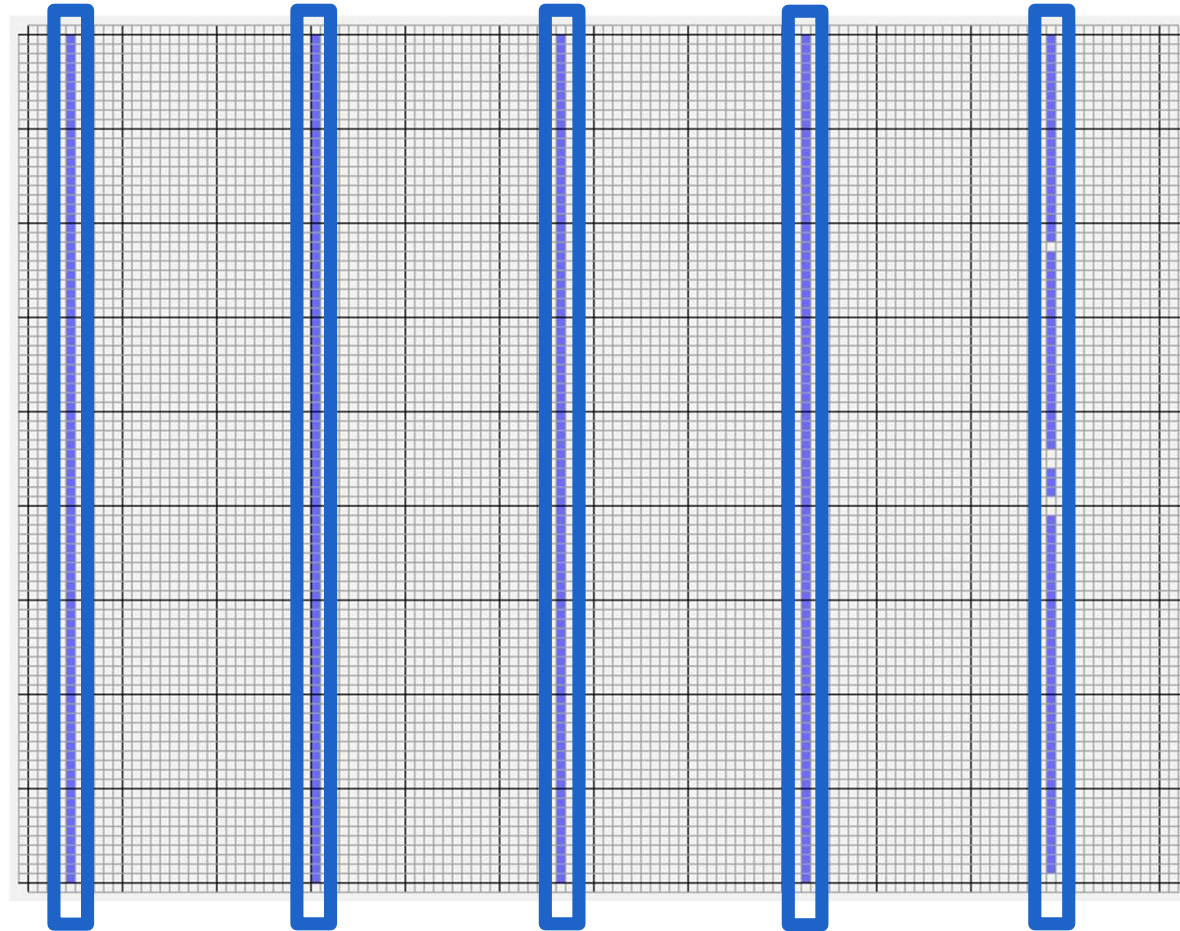
Hard Block Legalization

3. Greedy legalization



Hard Block Legalization

4. Simulated annealing-based optimization

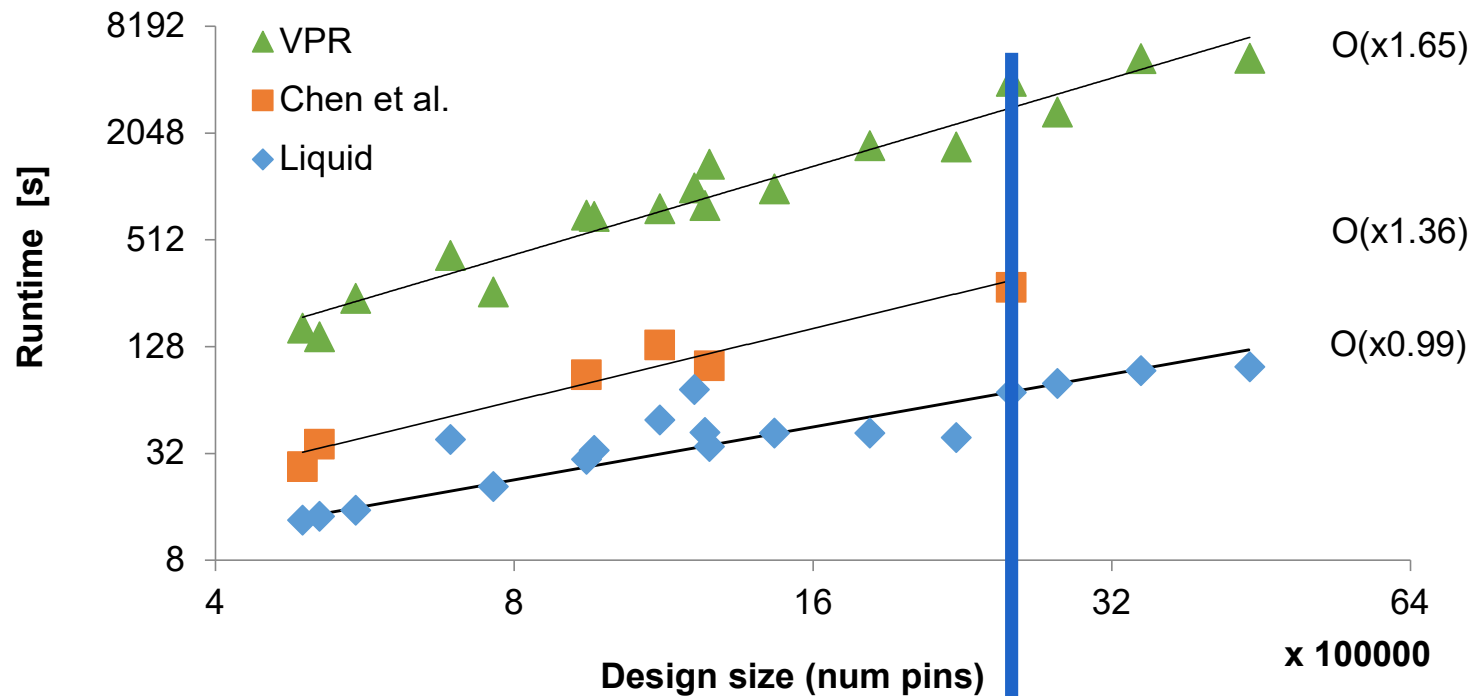


Liquid versus vpr

- Large heterogeneous Titan23 benchmark designs
- Model of Altera's Stratix IV FPGA

	Total wire-length [M]	Critical path delay [ns]	Runtime [s]
VPR	3.83	21.8	924
Liquid	3.85	21.7	39
ratio	1.00	0.99	0.04

scalability

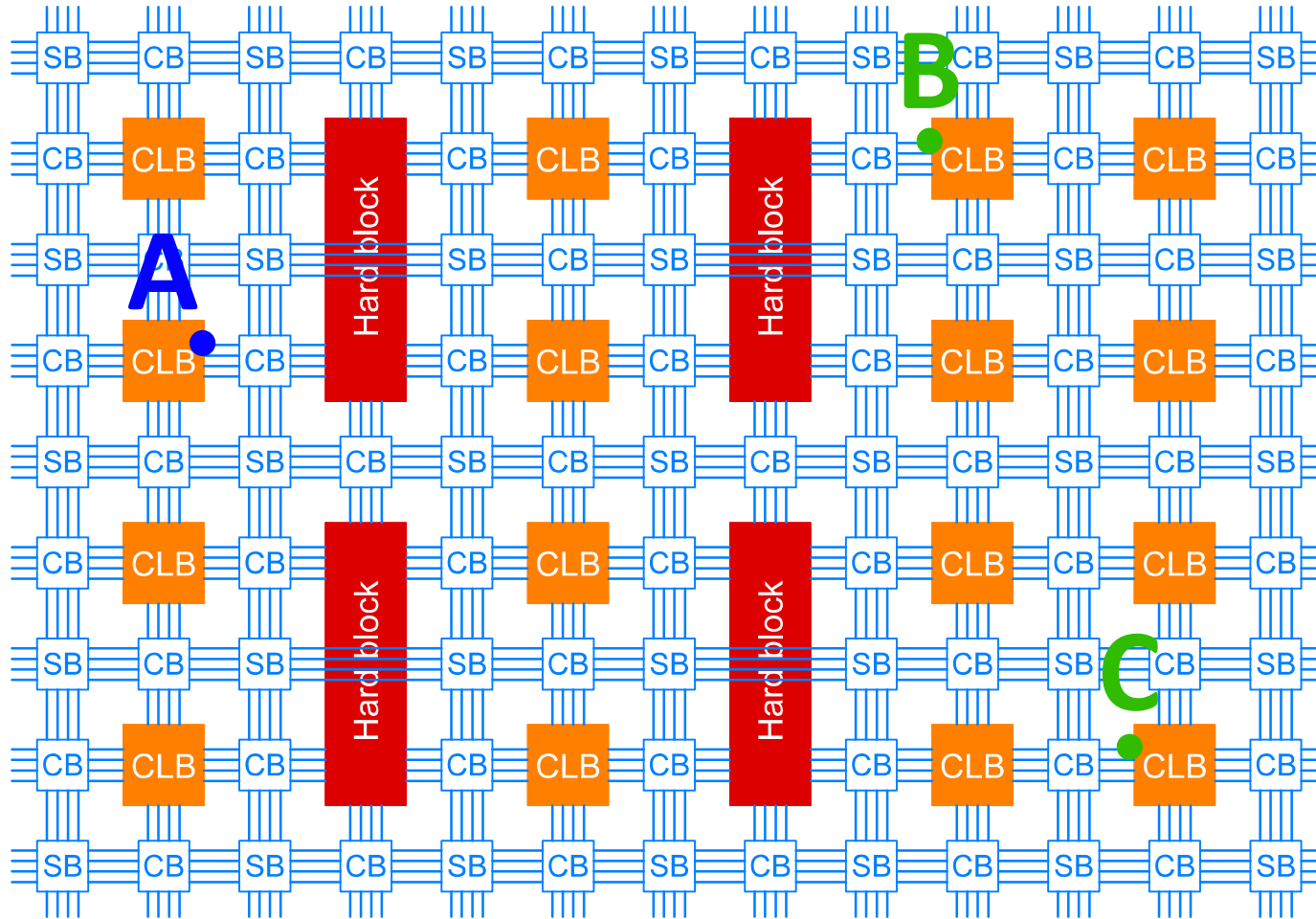


	VPR	4008 s		
denoise	Chen et al.	276 s	14.5x	
	Liquid	71 s	56.6x	3.9x

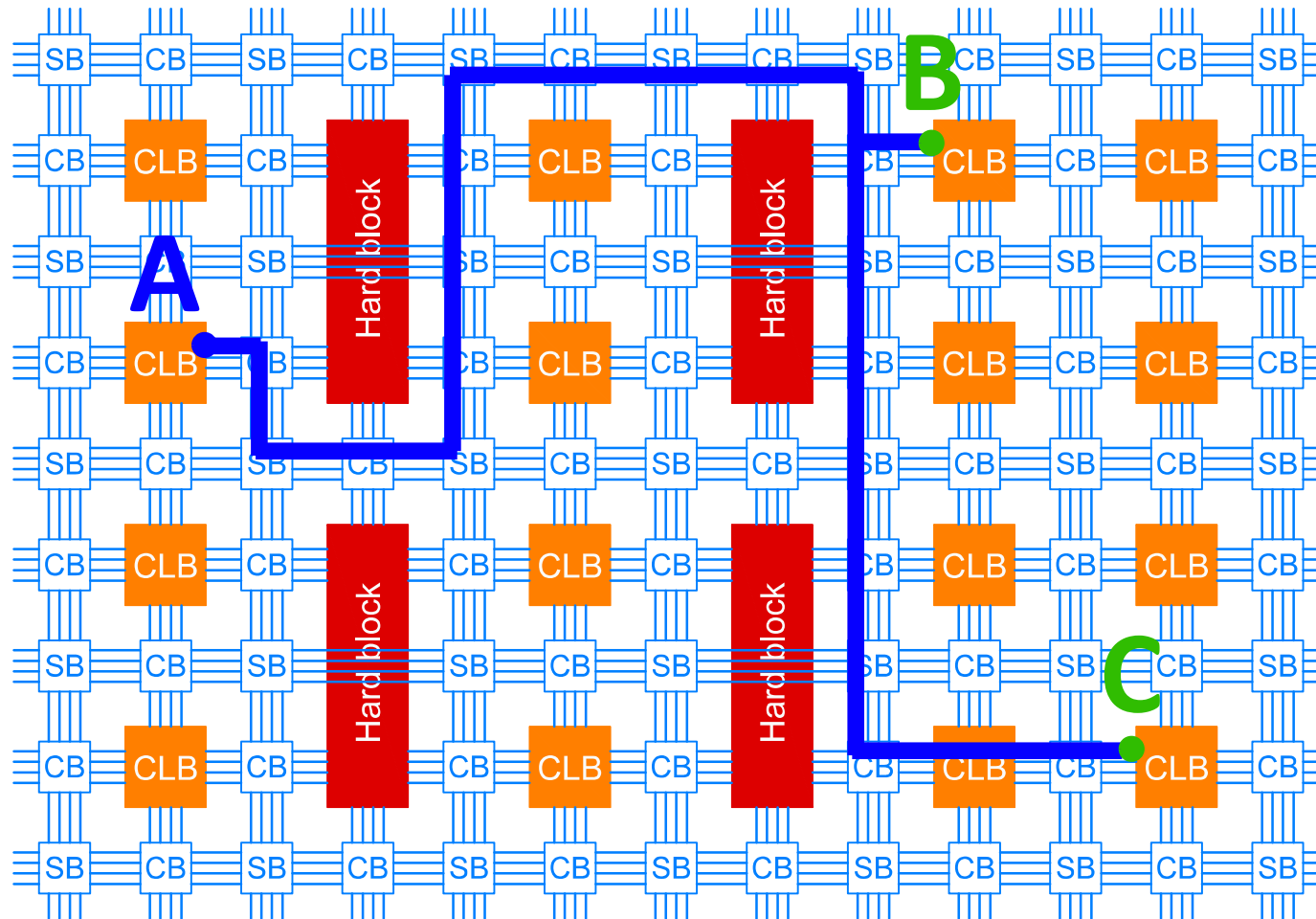
S.-Y. Chen and Y.-W. Chang, "Routing-architecture-aware analytical placement for heterogeneous FPGAs," in *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 2015, p. 27.

CRoute:
A Fast High-quality
Timing-driven
Connection-based
FPGA Router

Routing Problem



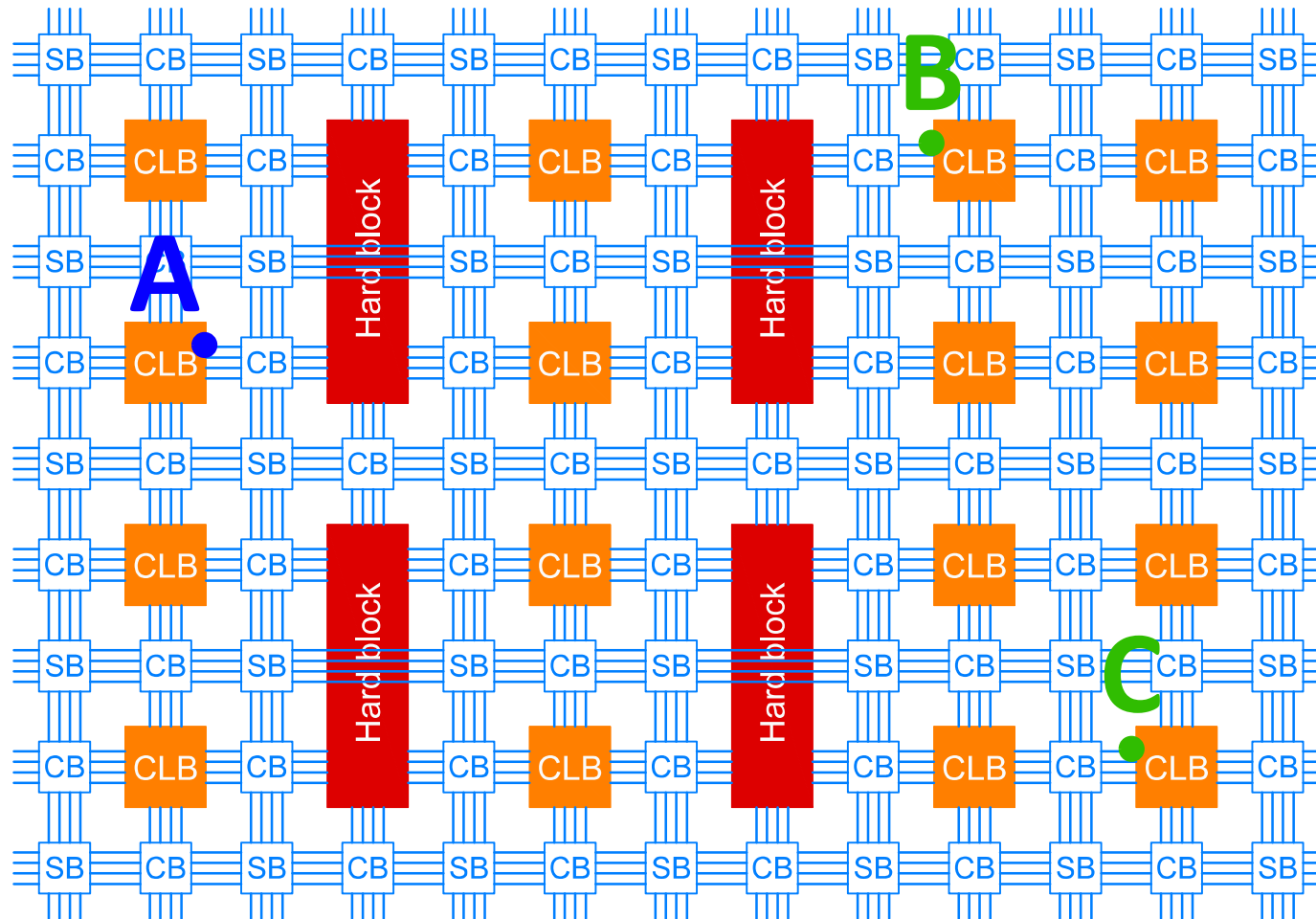
Routing Problem



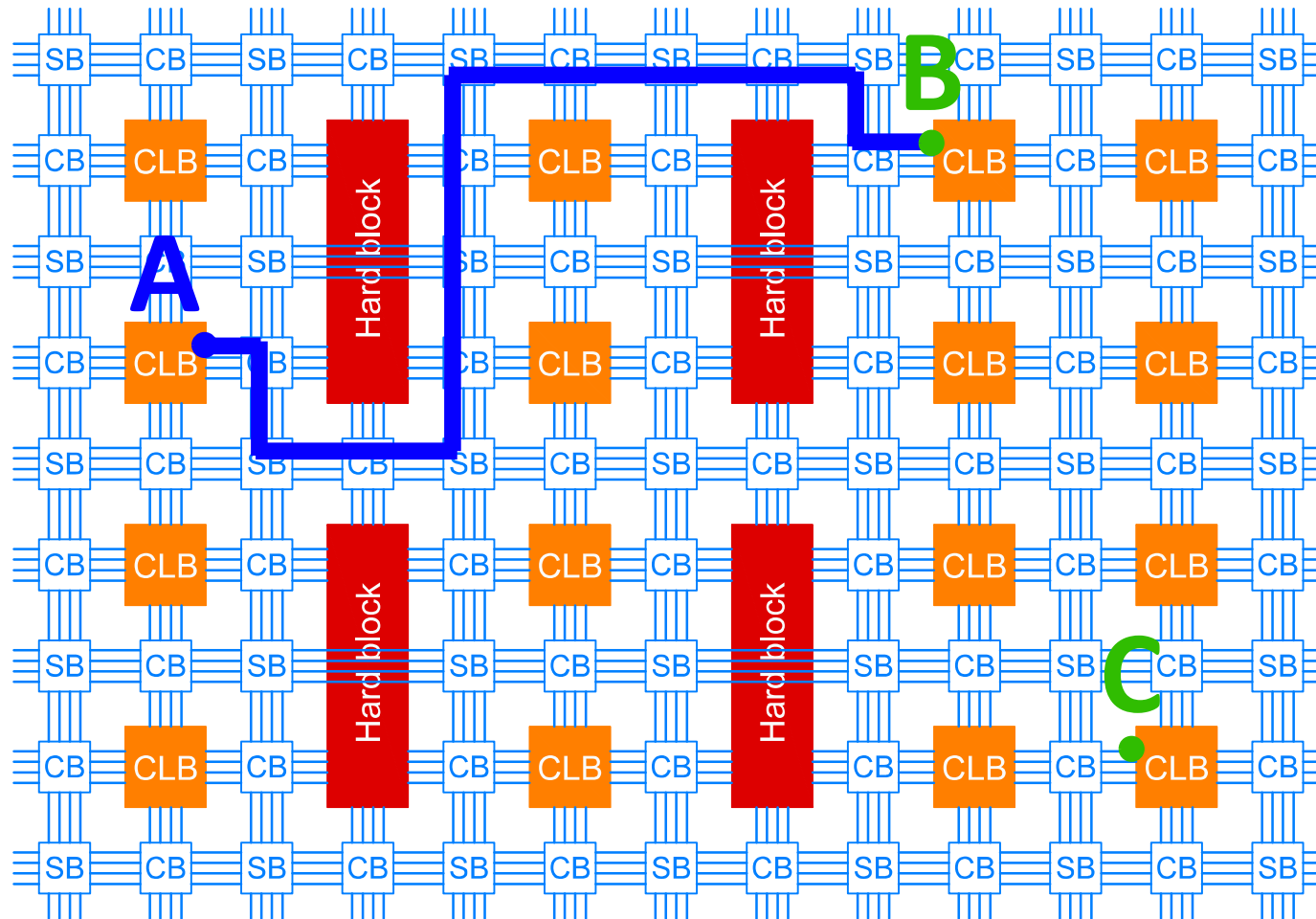
Net-based routing: Pathfinder

```
while (IllegalRoutingResourcesExist()):  
    for each Net n do:  
        if (firstIteration or n.congested()):  
            ripUpRouting(n)  
            route(n)  
            n.resources().updatePresentCongestionCost()  
  
    allResources().updateHistoryCost()  
    updatePresentCongestionMultiplier()  
    allResources().updatePresentCongestionCost()  
  
function route(Net n):  
    routingTree = {source}  
    for each Sink s of n:  
        path = Dijkstra(routingTree, s)  
        routingTree = routingTree U path
```

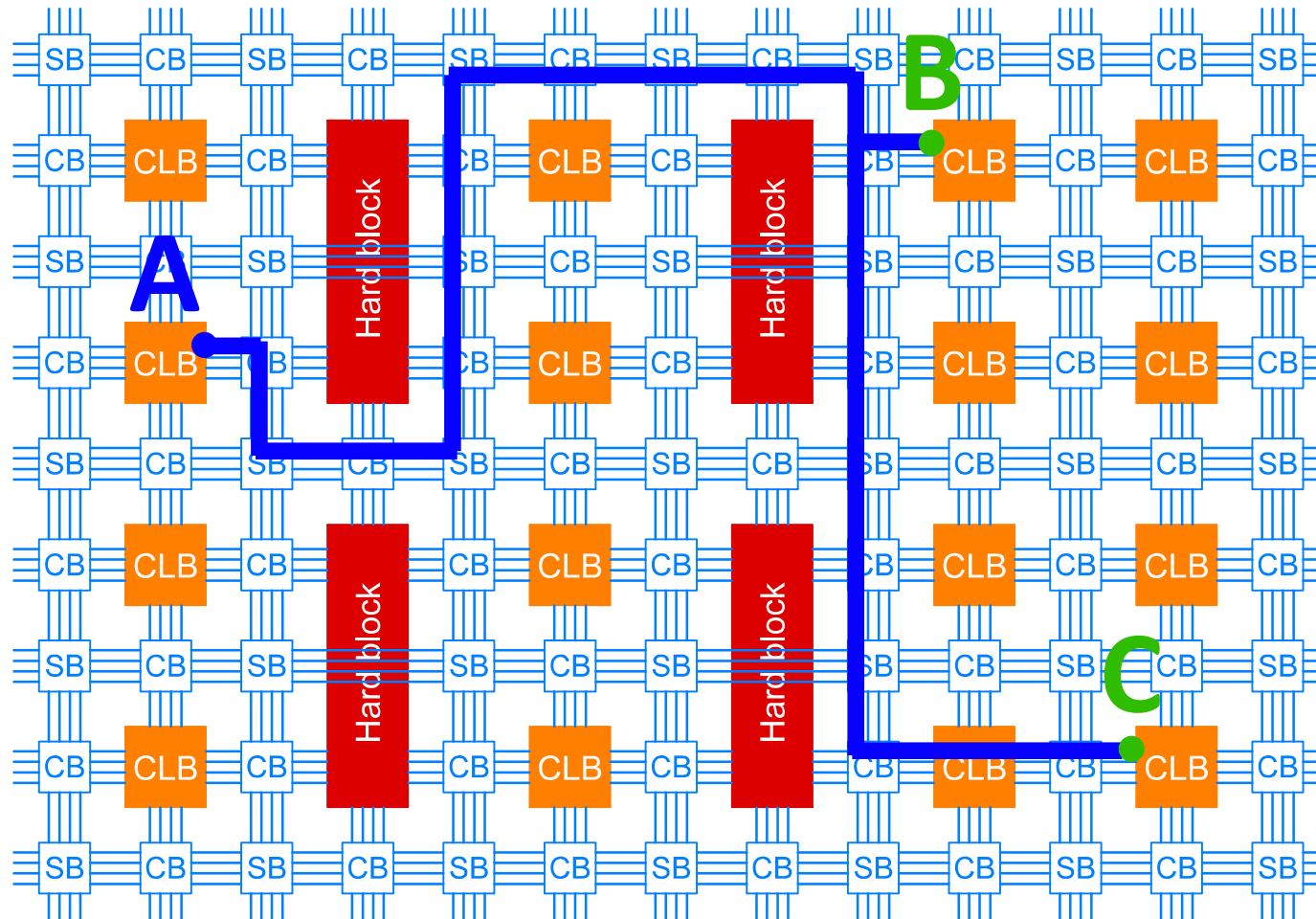
Routing Problem



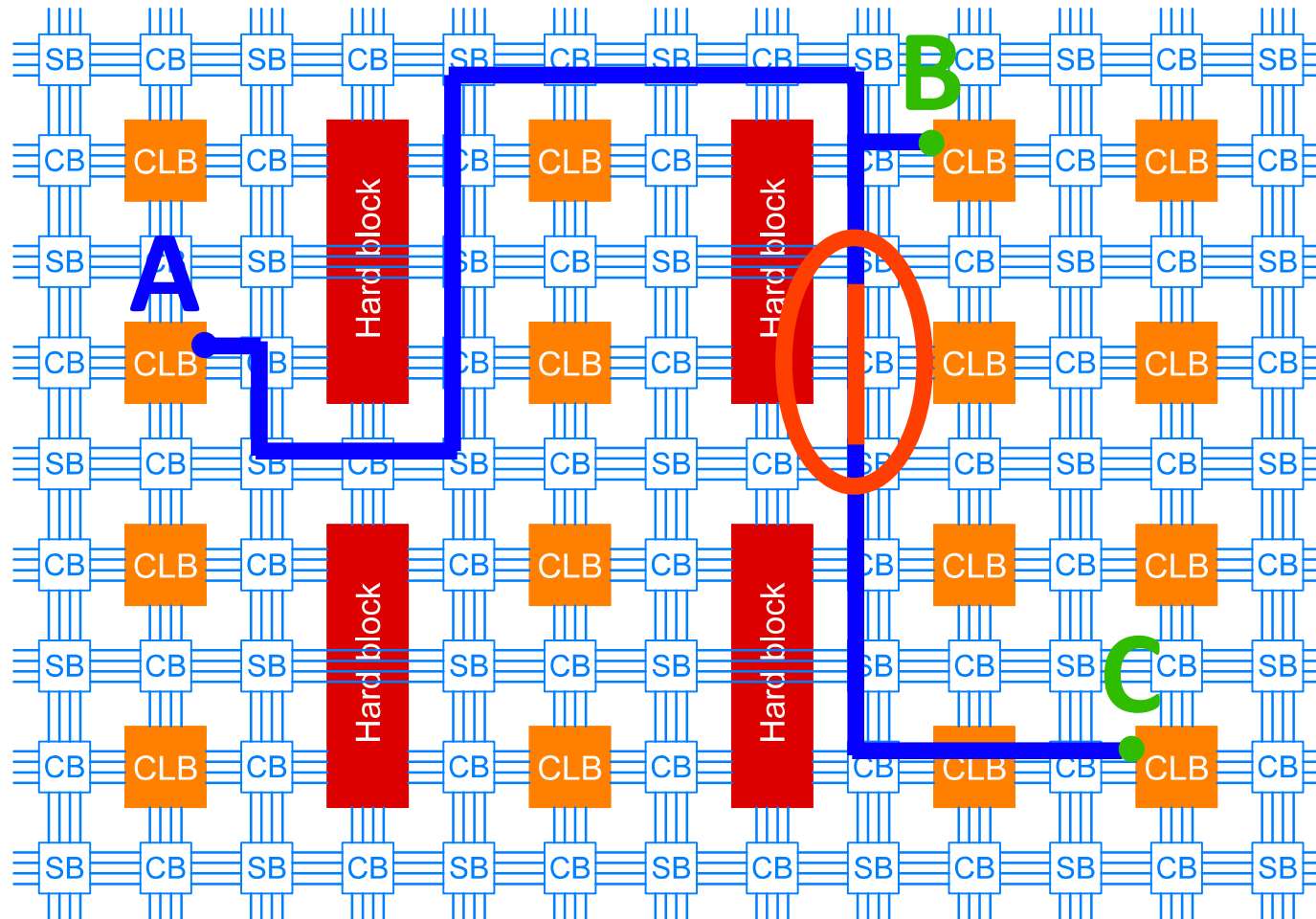
Routing Problem



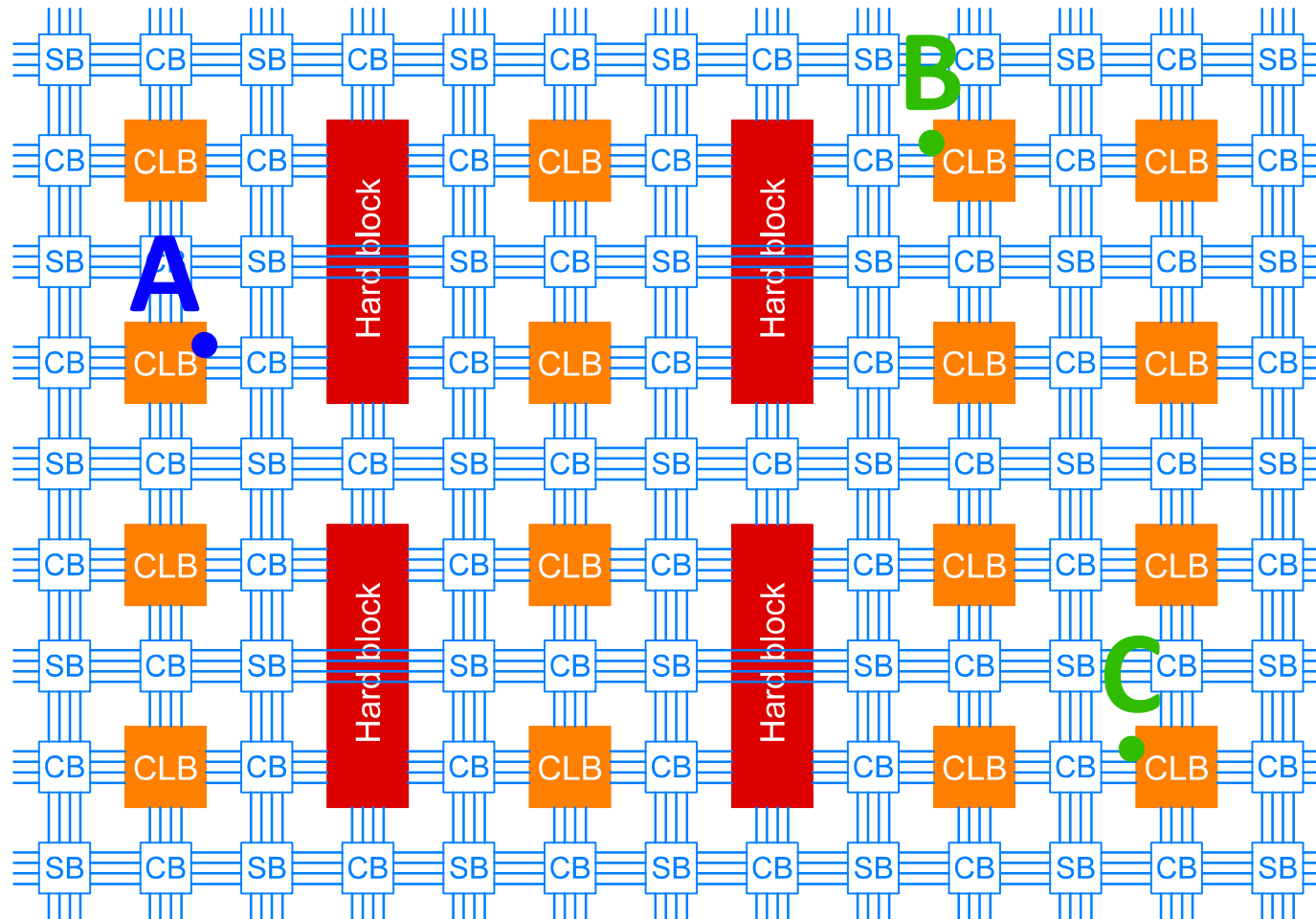
Routing Problem



Routing Problem



Routing Problem

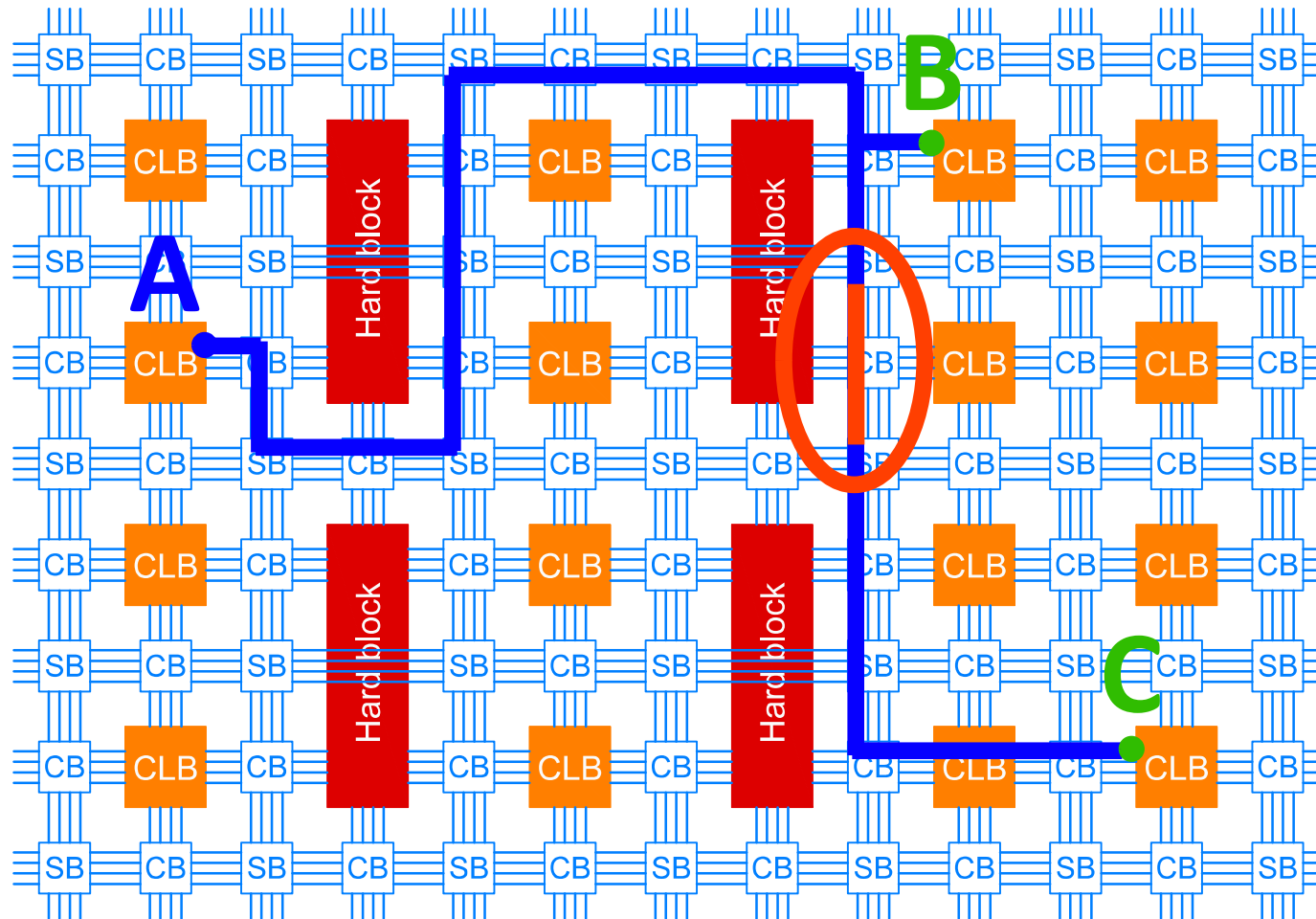


connection-based routing

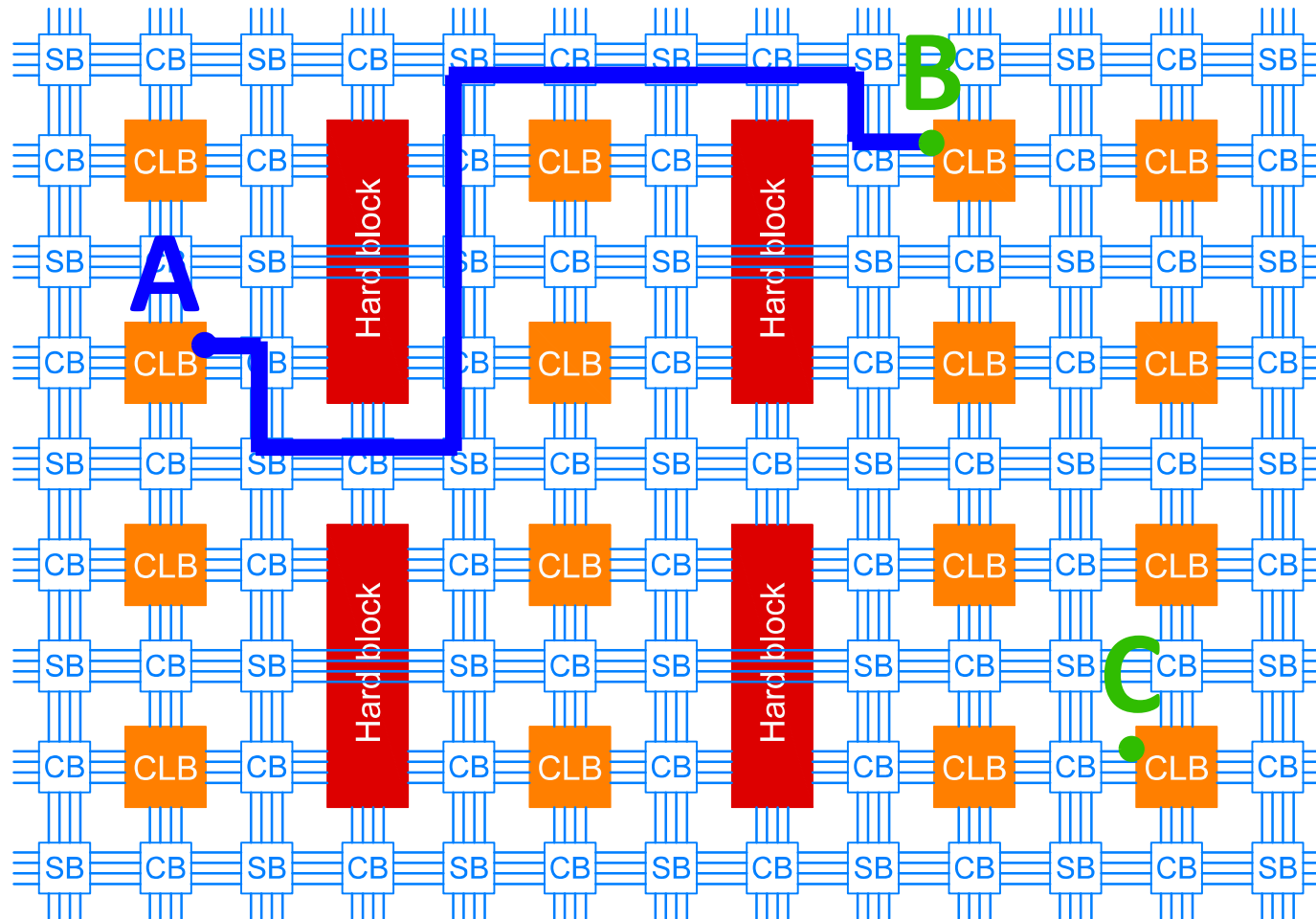
```
while (IllegalRoutingResourcesExist()):  
    for each Conn c do:  
        if (firstIteration or c.congested()):  
            ripUpRouting(c)  
            route(c)  
            c.resources().updatePresentCongestionCost()  
  
allResources().updateHistoryCost()  
updatePresentCongestionMultiplier()  
allResources().updatePresentCongestionCost()
```

E. Vansteenkiste, K. Bruneel, and D. Stroobandt, "A connection-based router for FPGAs," in *Field-Programmable Technology (FPT), 2013 International Conference on*. IEEE, 2013, pp. 326–329.

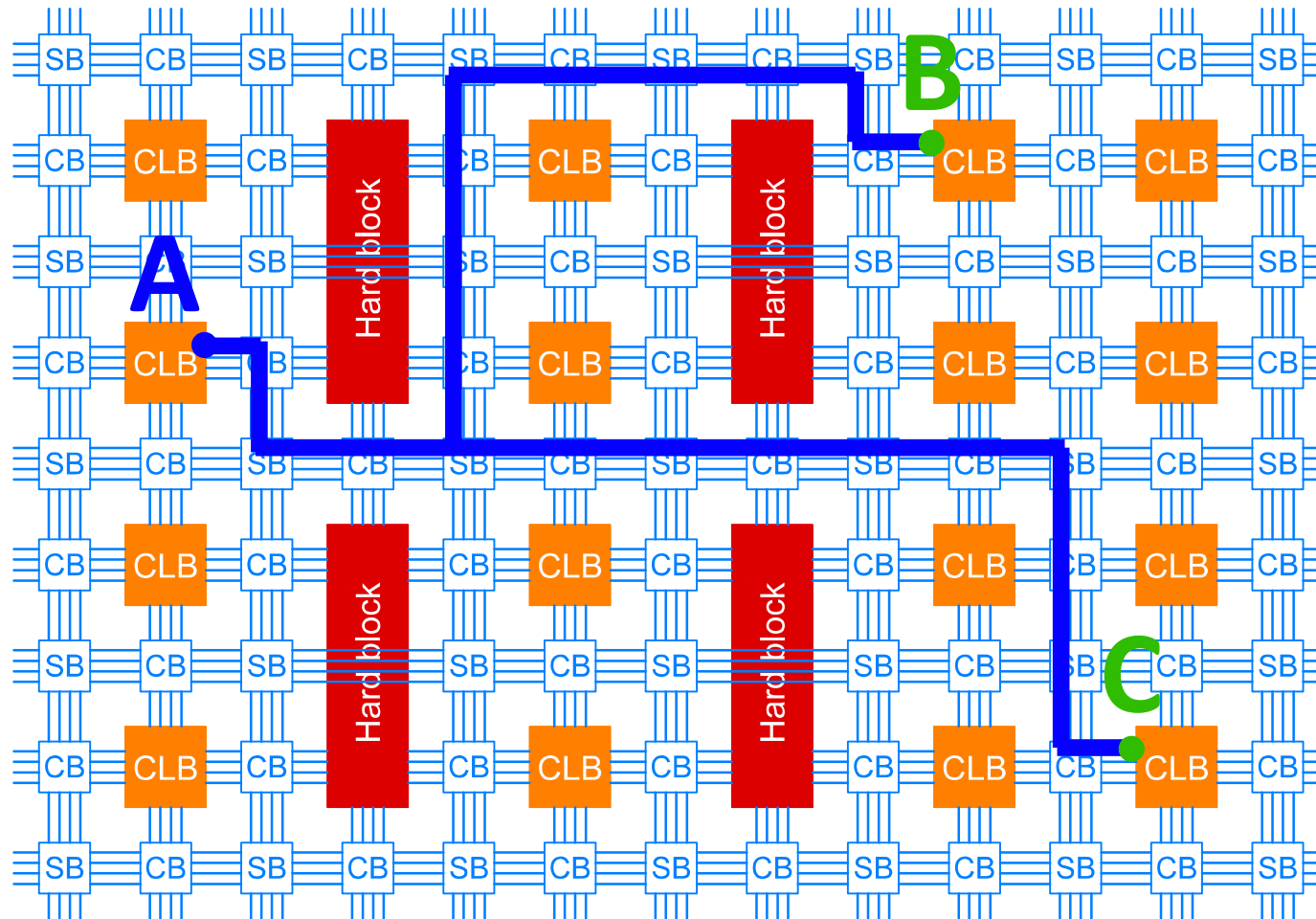
Routing Problem



Routing Problem



Routing Problem

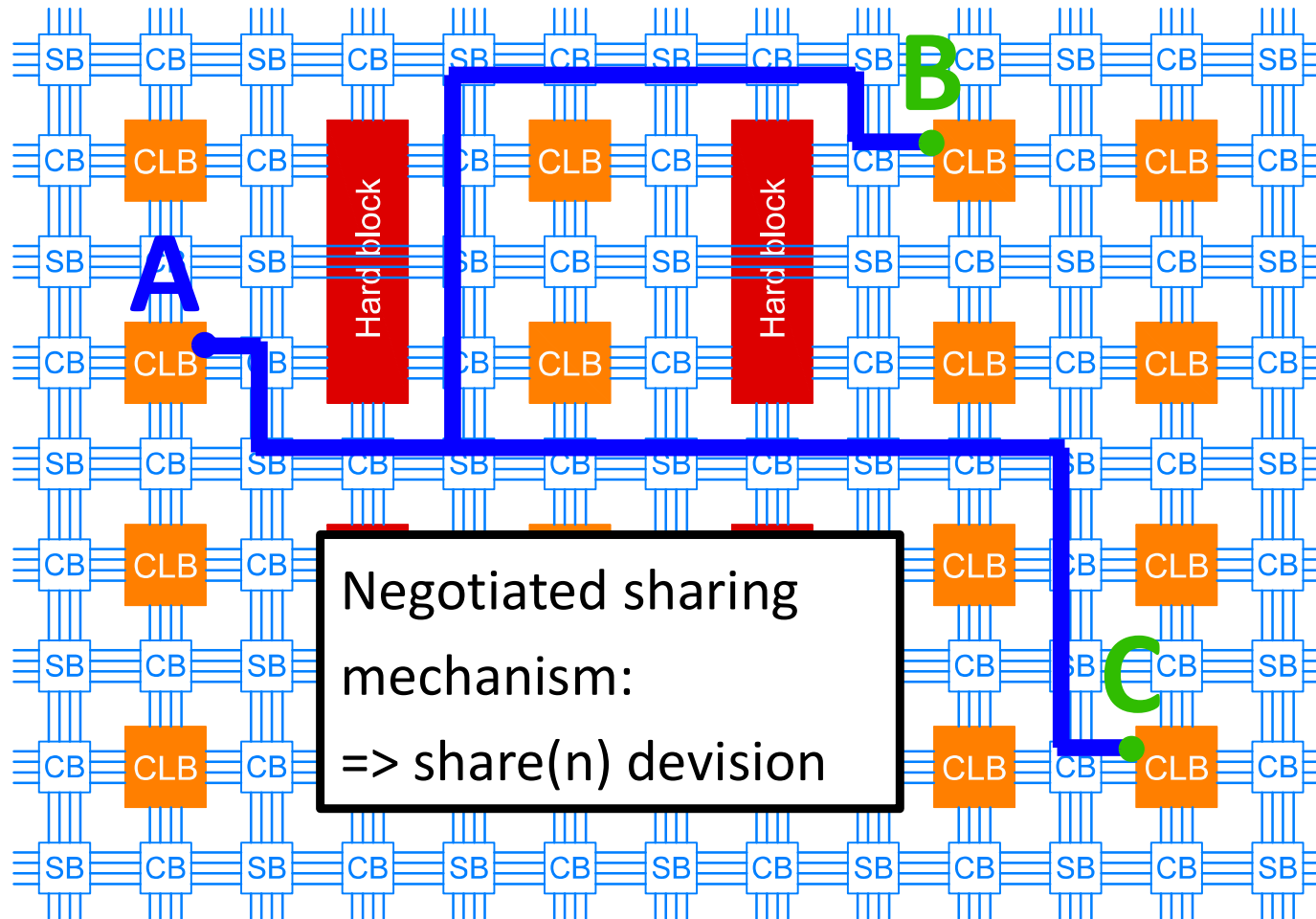


connection-based routing

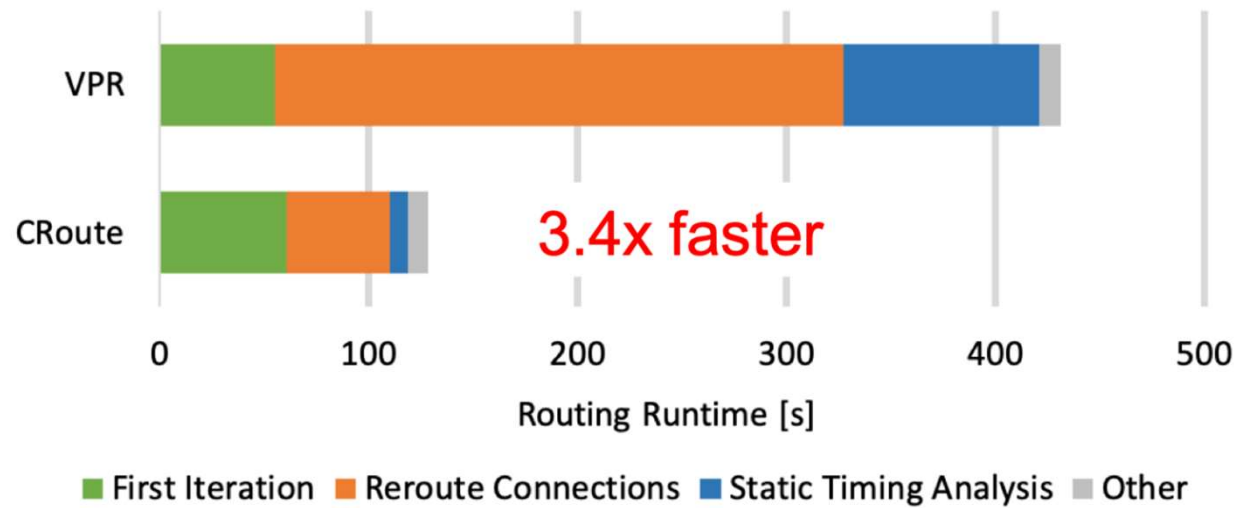
```
while (IllegalRoutingResourcesExist()):  
    for each Conn c do:  
        if (firstIteration or c.congested()):  
            ripUpRouting(c)  
            route(c)  
            c.resources().updatePresentCongestionCost()  
  
    allResources().updateHistoryCost()  
    updatePresentCongestionMultiplier()  
    allResources().updatePresentCongestionCost()  
  
function route(Conn c):  
    path = Dijkstra(c.getSink())
```

E. Vansteenkiste, K. Bruneel, and D. Stroobandt, "A connection-based router for FPGAs," in *Field-Programmable Technology (FPT), 2013 International Conference on*. IEEE, 2013, pp. 326–329.

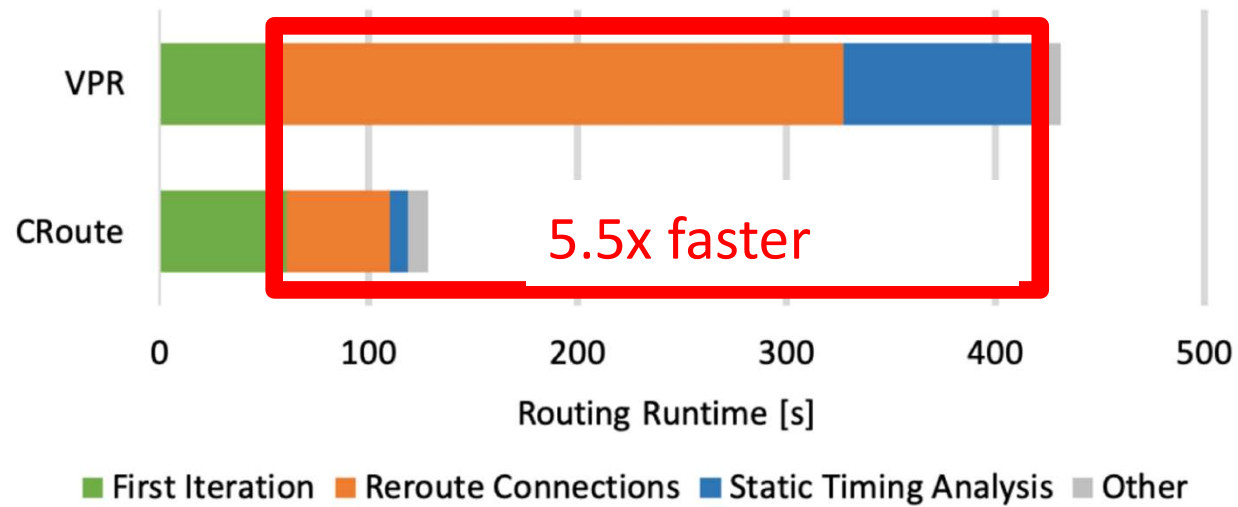
Routing Problem



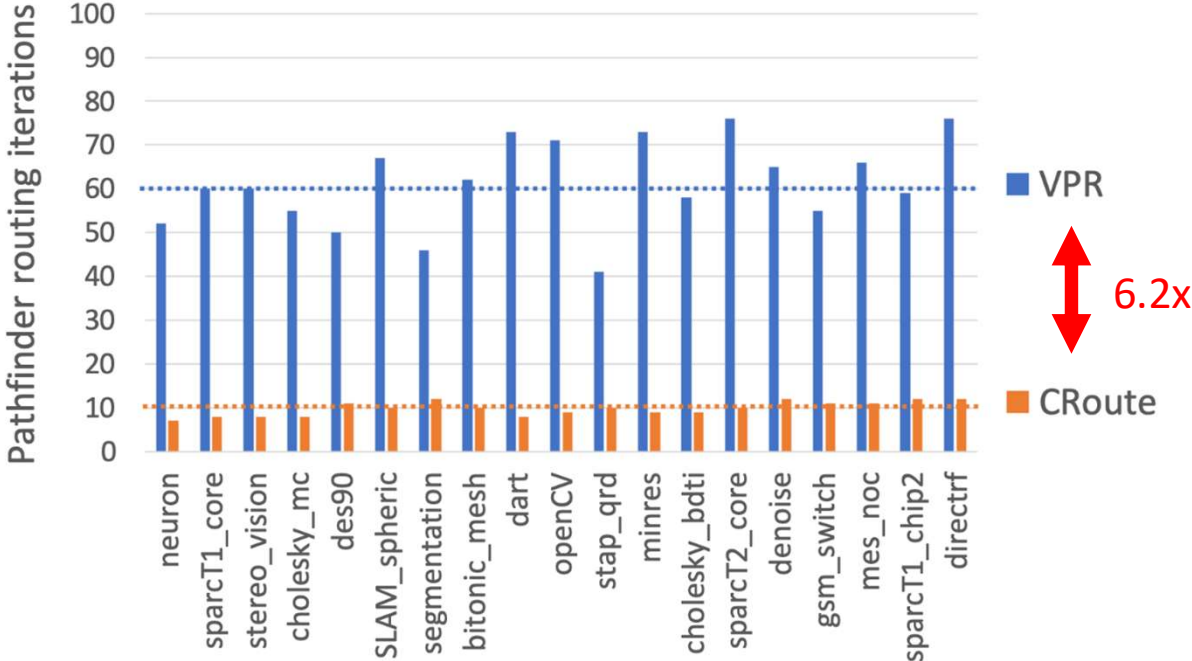
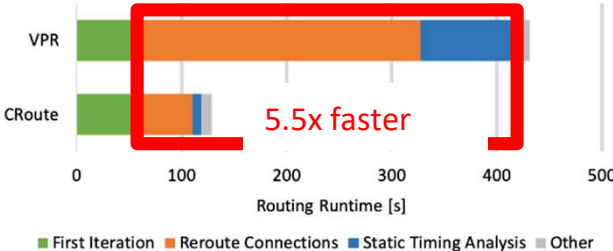
Runtime



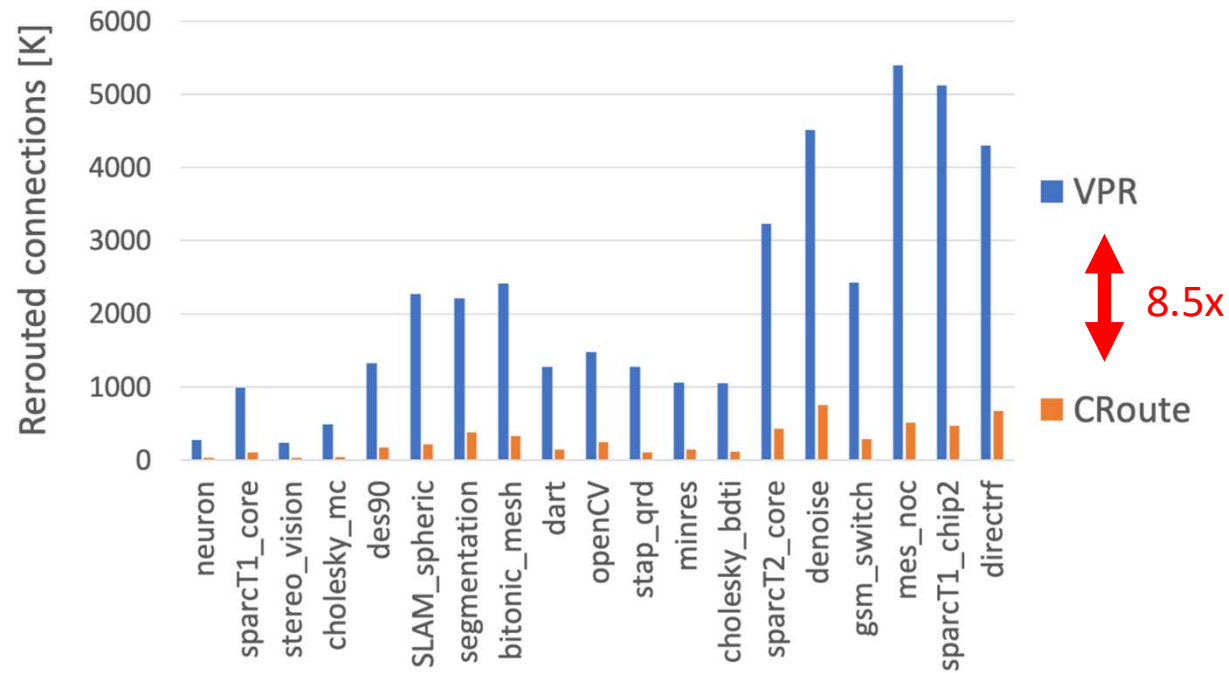
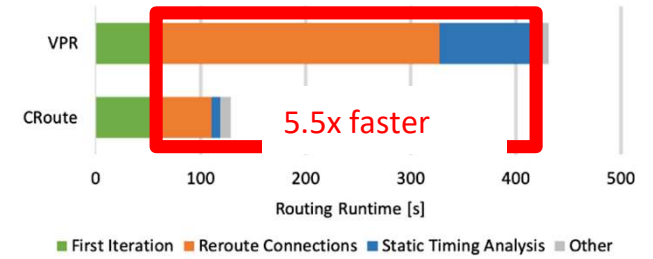
Runtime



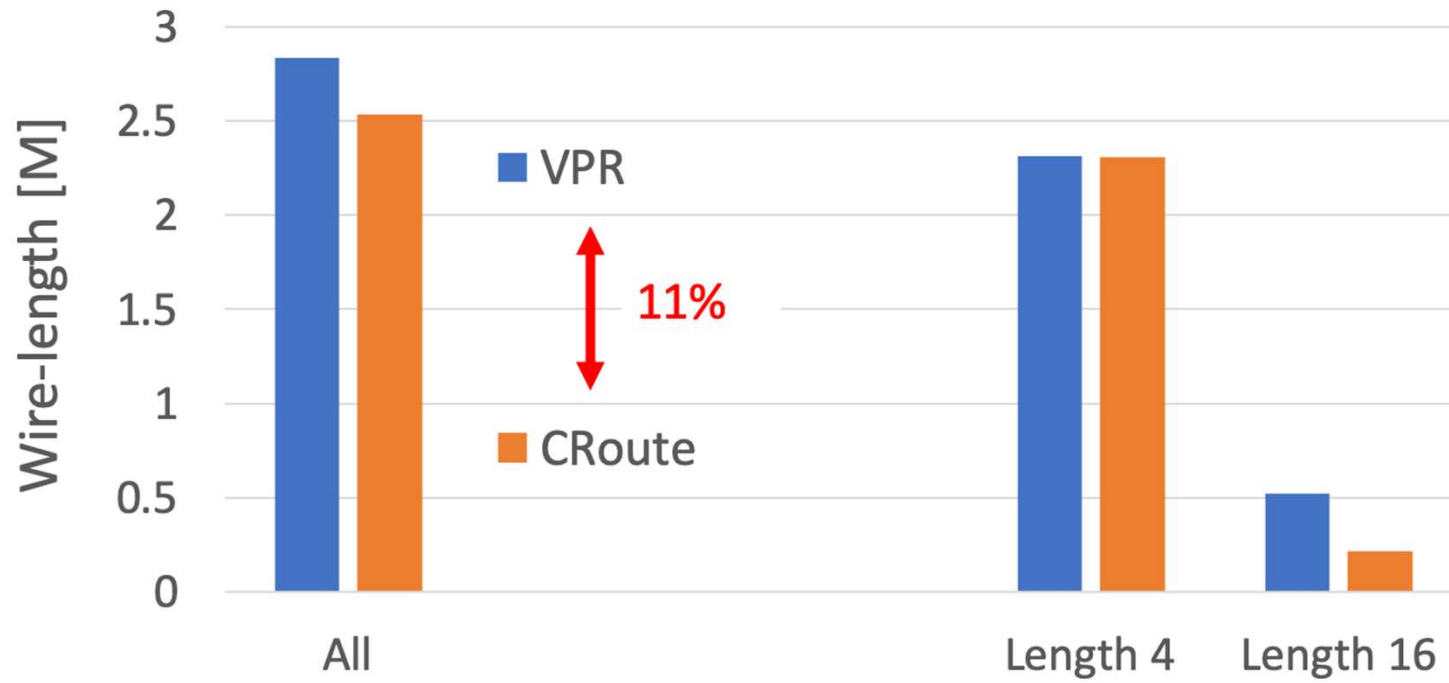
runtime



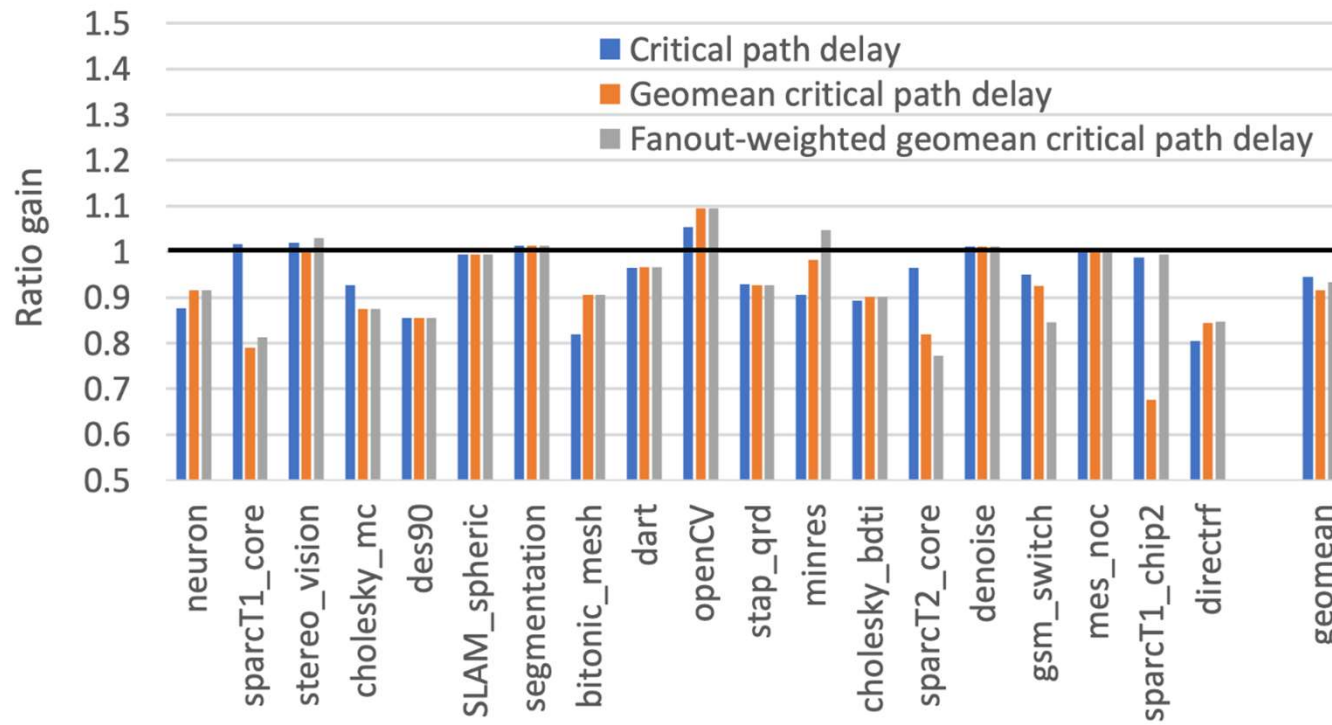
Runtime



Wire-length

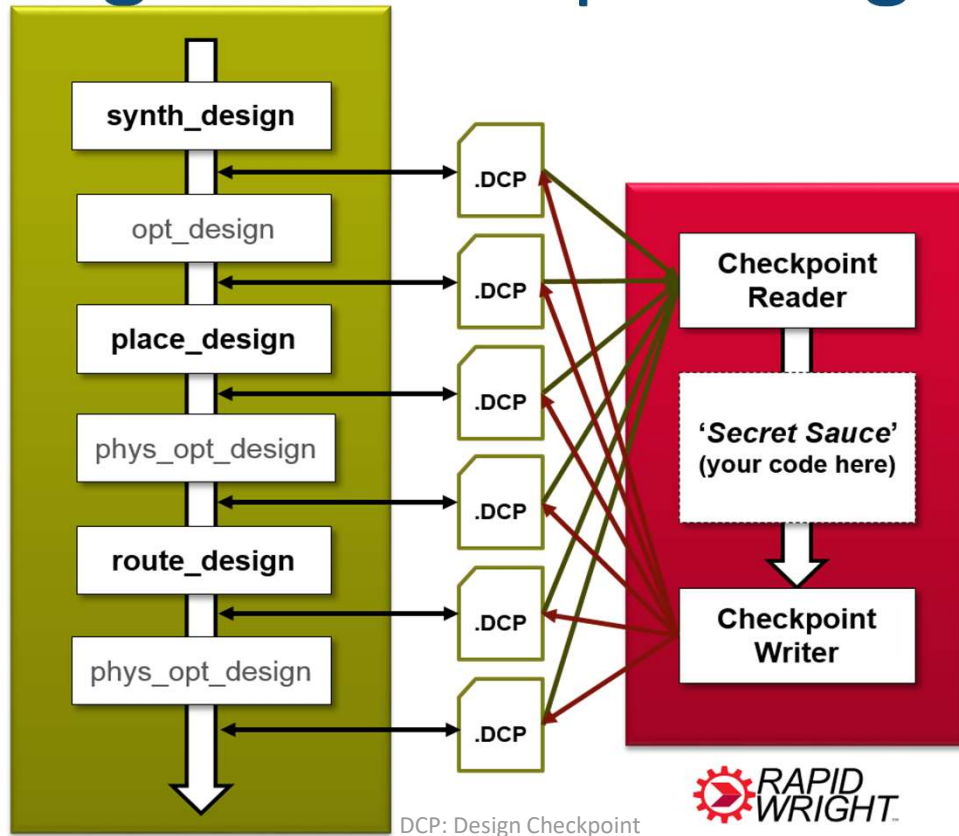


Critical path delay



RWRoute: An Open-source Timing-driven Router for Commercial FPGAs

Background: RapidWright



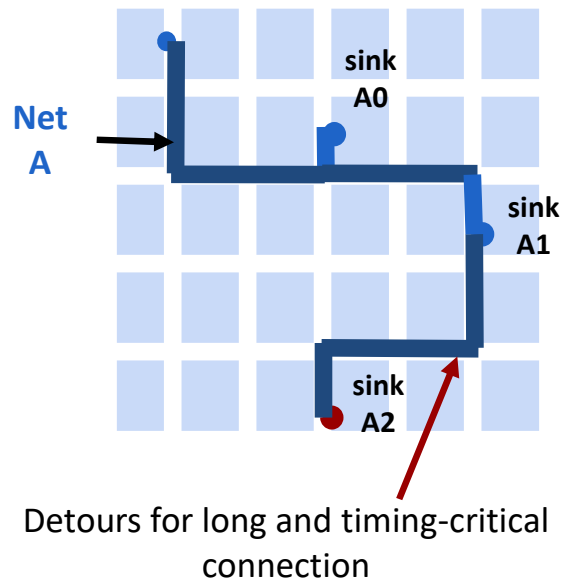
VIVADO

Source:

Chris Lavin and Alireza Kaviani, RapidWright: Enabling Custom Crafted Implementations for FPGAs, FCCM 2018.

- Companion open-source framework for Vivado
- Enables custom crafted implementations
- Enables targeting commercial FPGAs

CRoute: Sharing Mechanism and Drawback



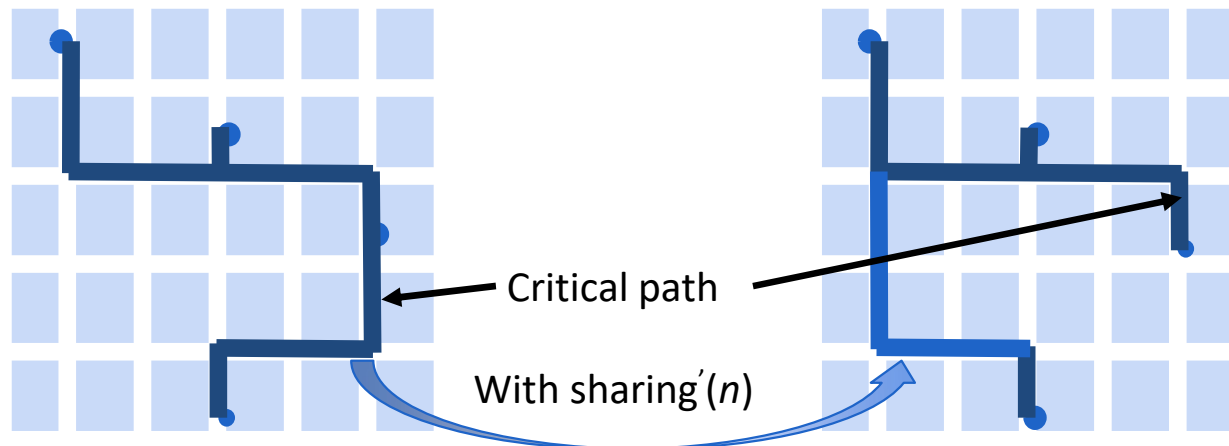
- Scales down the cost of using a node n with a sharing factor, i.e., $\text{sharing}(n)$
- $\text{sharing}(n) = \# \text{ connections using the node } n$
 - Unaware of the criticality of connection under consideration
 - Encourages resource sharing even when a connection is long and timing-critical
 - Limits critical path delay optimization

Source:

Elias Vansteenkiste, Karel Bruneel and D. Stroobandt, A connection-based router for FPGAs, FPT 2013.

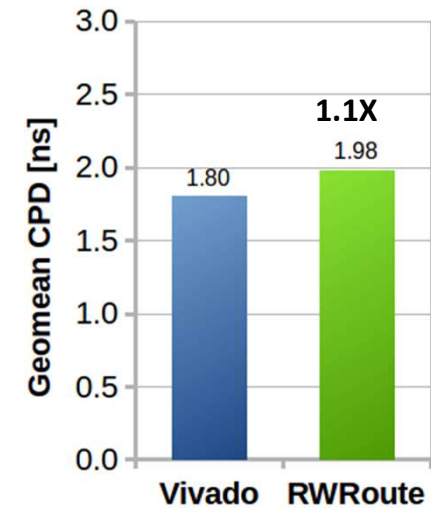
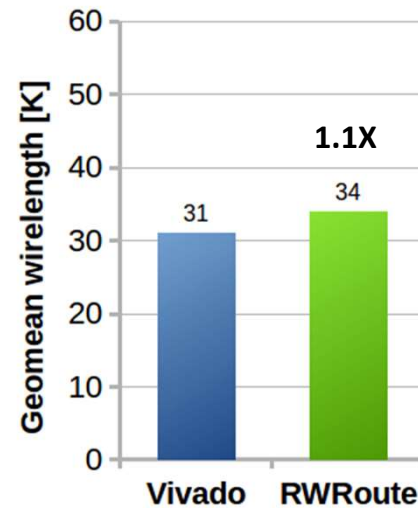
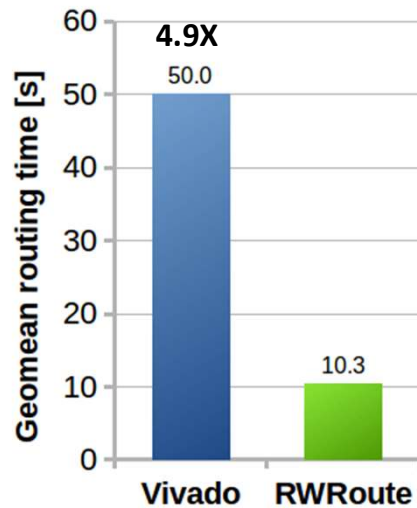
RWRoute: Criticality-aware Sharing Mechanism

- $\text{sharing}'(n) = (1 - \text{criticality})^\lambda \times \text{sharing}(n)$
 - λ : user-defined sharing exponent
 - $\lambda = 0$: the criticality-unaware sharing mechanism as that of CRoute
 - $\lambda \geq 1$: effective criticality-aware sharing mechanism
 - Encourage timing optimization for critical connections

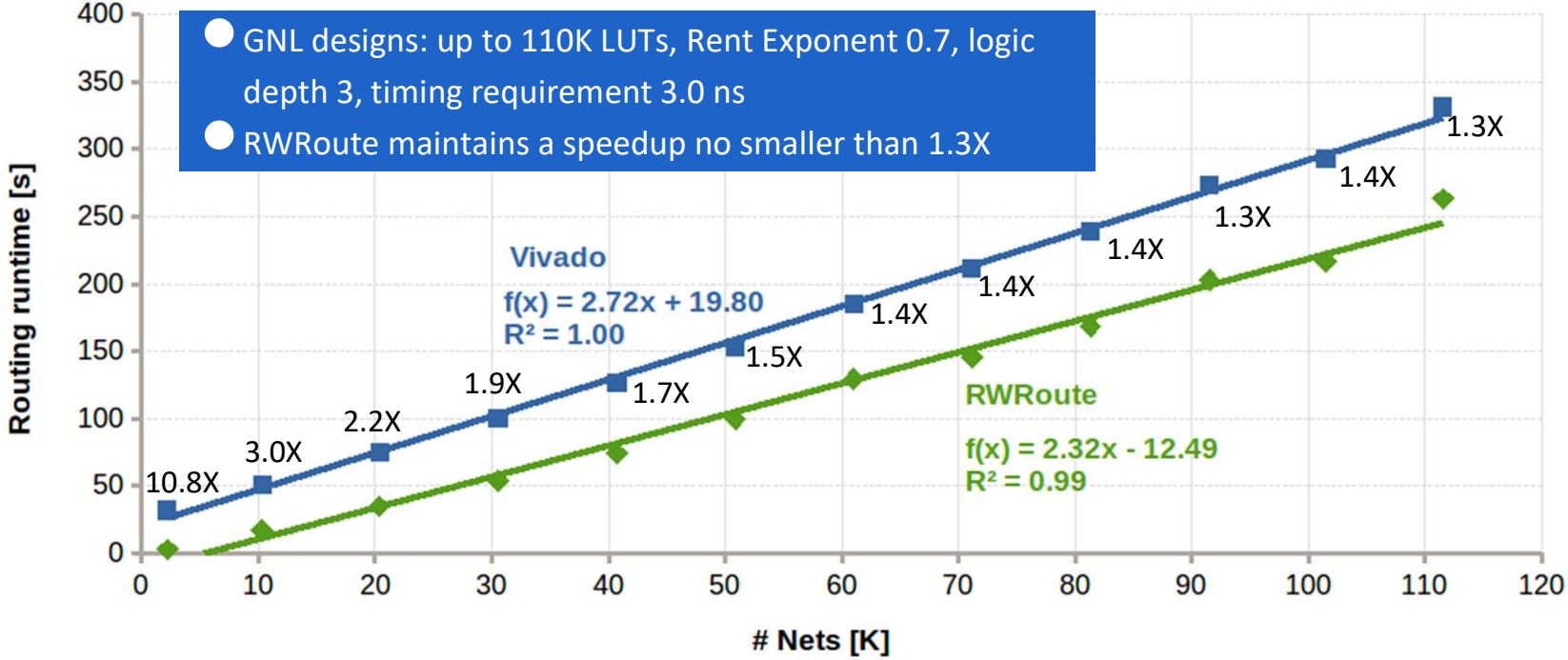


Runtime and QoR Comparisons with Vivado

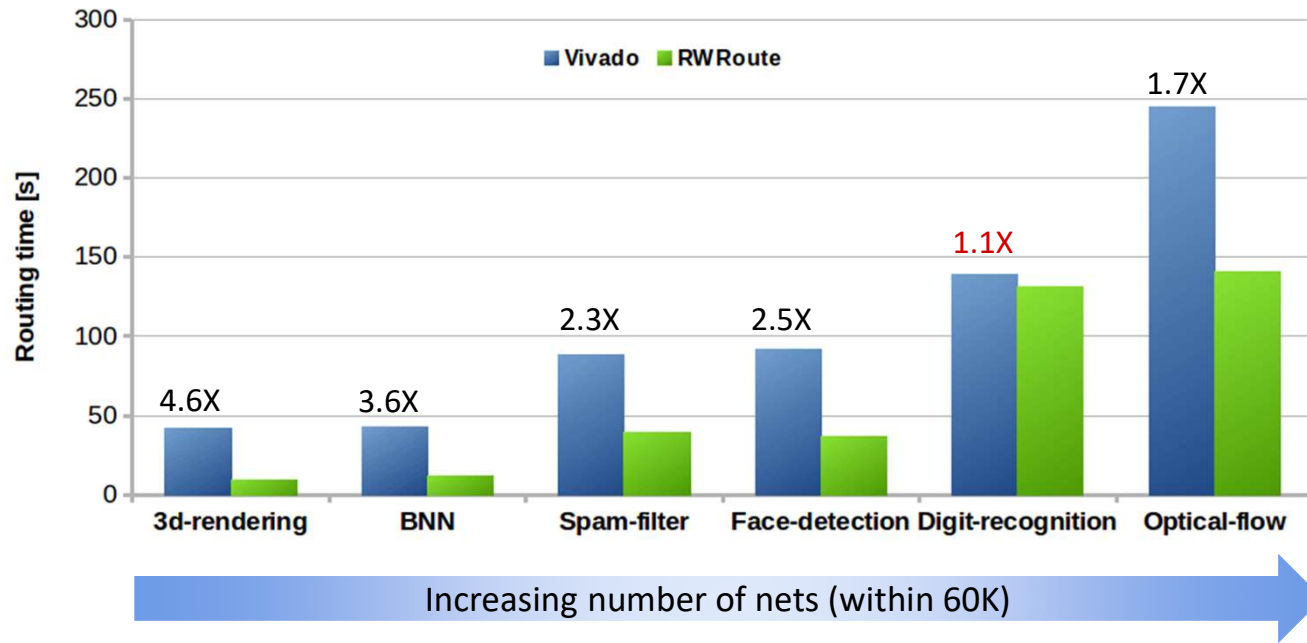
- 4.9X runtime speedup
- 10% longer wirelength
- 10% longer critical path delay



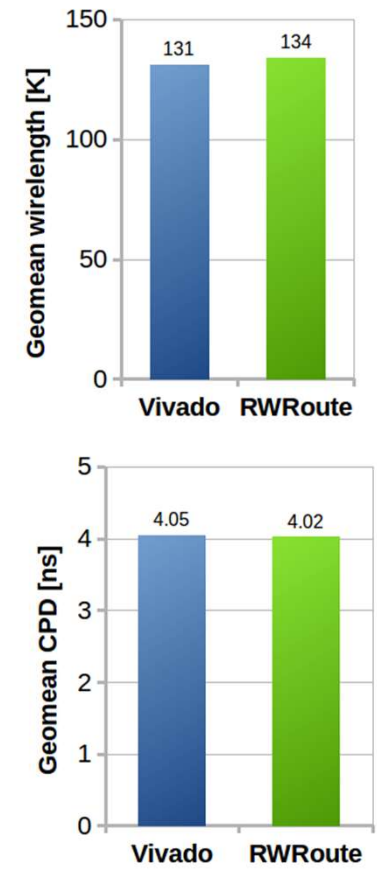
Scalability Comparison with Vivado



Comparisons Regarding Rosetta Benchmarks



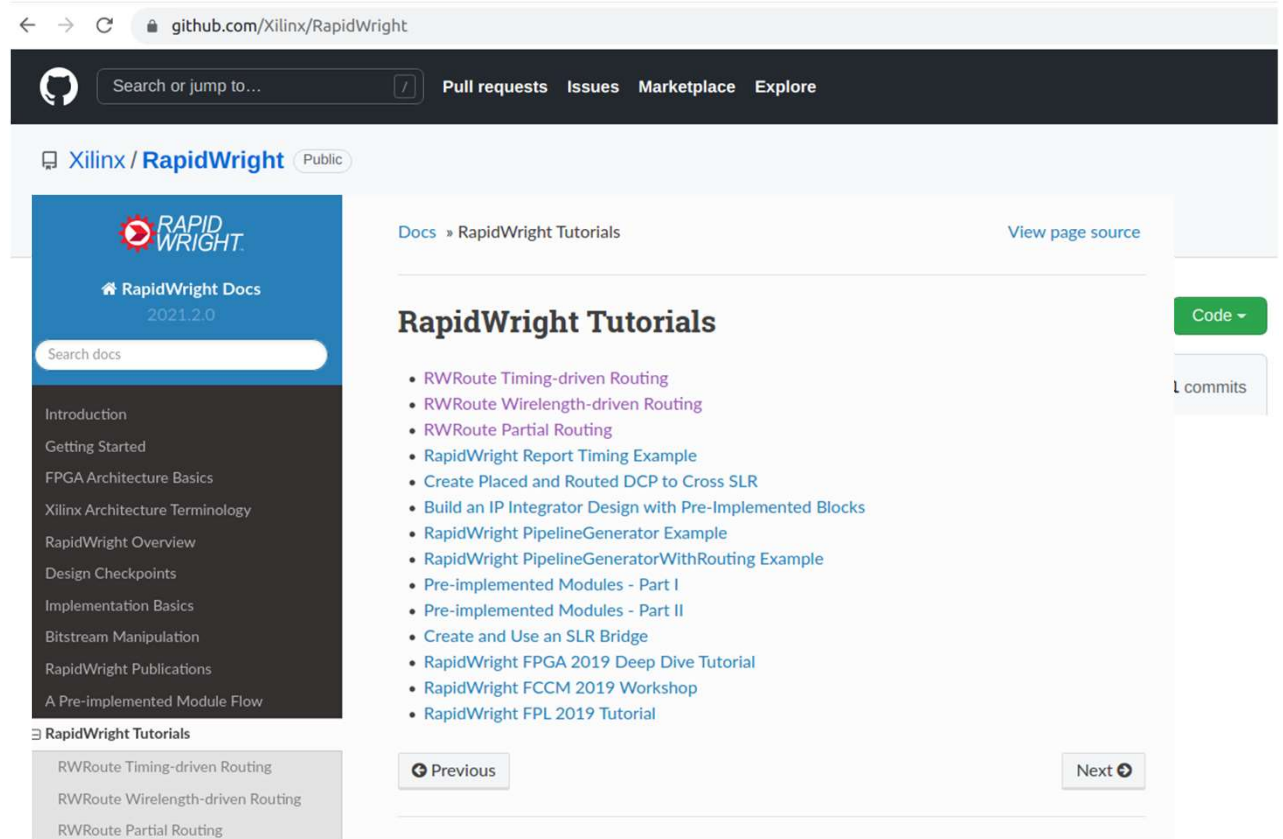
- 1.1X ~ 4.6X runtime speedup
- Comparable QoR



How Do You Use RWRoute?



www.rapidwright.io



github.com/Xilinx/RapidWright

Search or jump to... Pull requests Issues Marketplace Explore

Xilinx / RapidWright Public

Docs » RapidWright Tutorials View page source

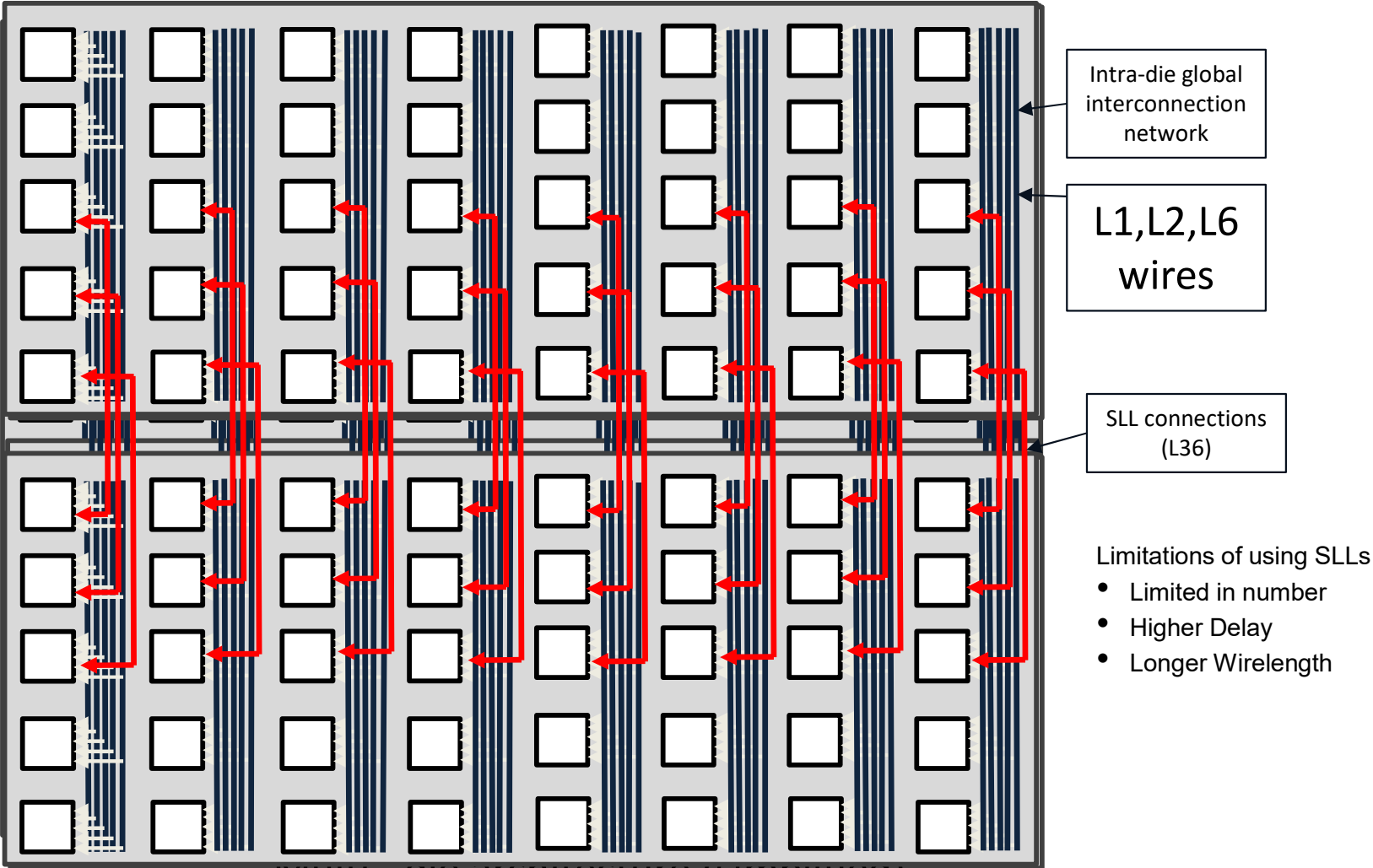
RapidWright Tutorials

- [RWRoute Timing-driven Routing](#)
- [RWRoute Wirelength-driven Routing](#)
- [RWRoute Partial Routing](#)
- [RapidWright Report Timing Example](#)
- [Create Placed and Routed DCP to Cross SLR](#)
- [Build an IP Integrator Design with Pre-Implemented Blocks](#)
- [RapidWright PipelineGenerator Example](#)
- [RapidWright PipelineGeneratorWithRouting Example](#)
- [Pre-implemented Modules - Part I](#)
- [Pre-implemented Modules - Part II](#)
- [Create and Use an SLR Bridge](#)
- [RapidWright FPGA 2019 Deep Dive Tutorial](#)
- [RapidWright FCCM 2019 Workshop](#)
- [RapidWright FPL 2019 Tutorial](#)

Previous Next

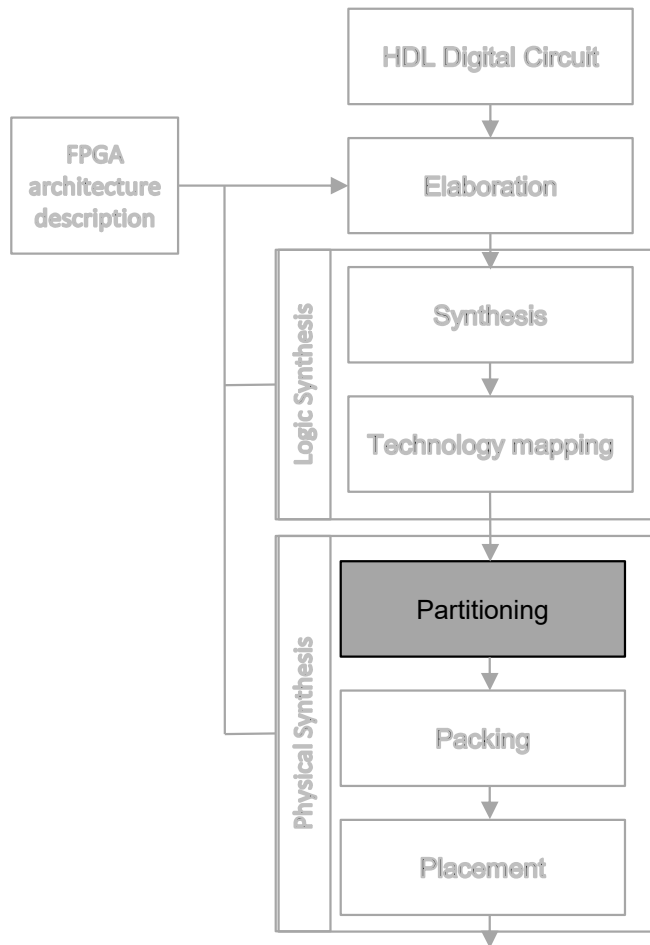
Multi-die placement and routing

Introduction



Single die architecture (Simplified)

Partitioning



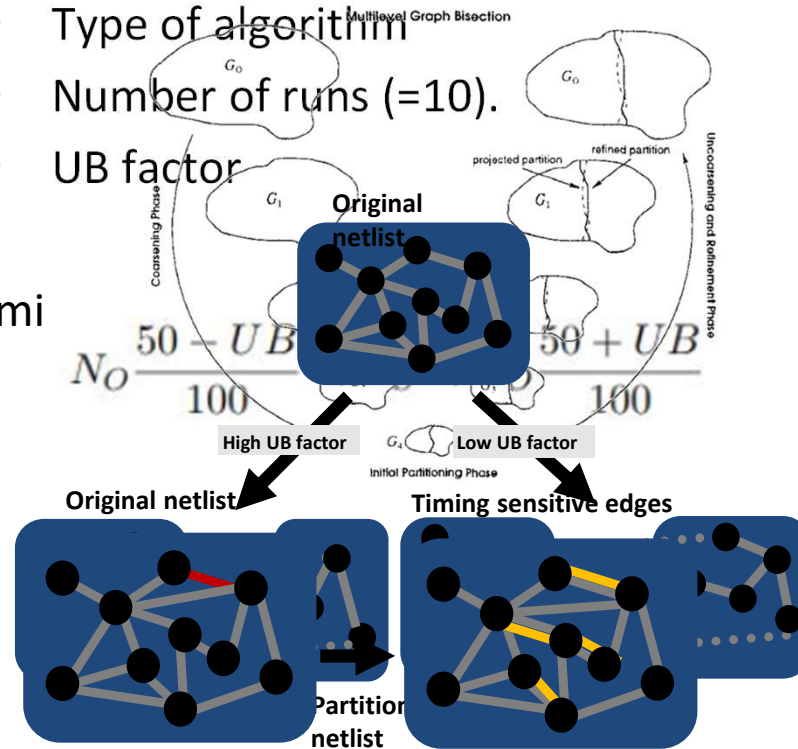
- Partitioning using hMETIS
- Factors influencing the partitioning quality.

- Type of algorithm

- Number of runs (=10).

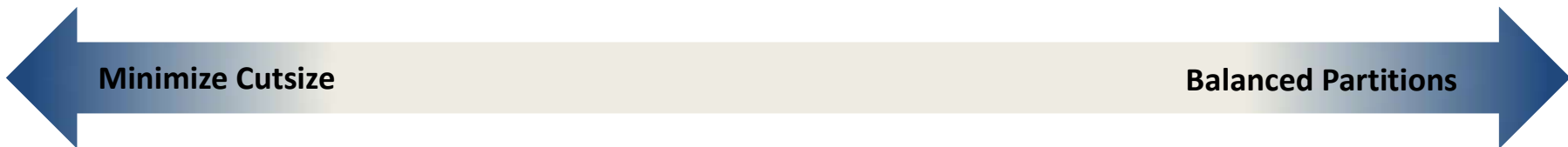
- UB factor

- Timi



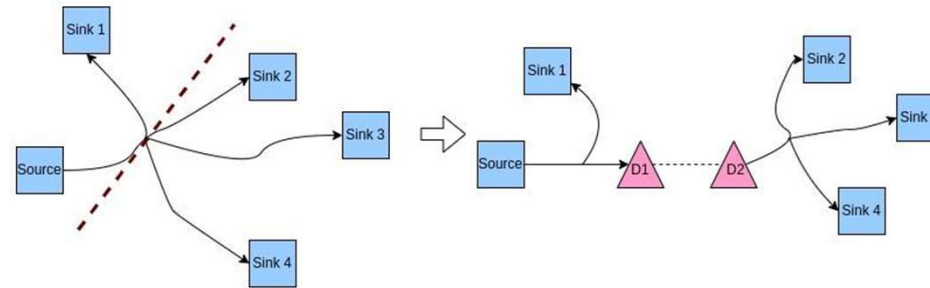
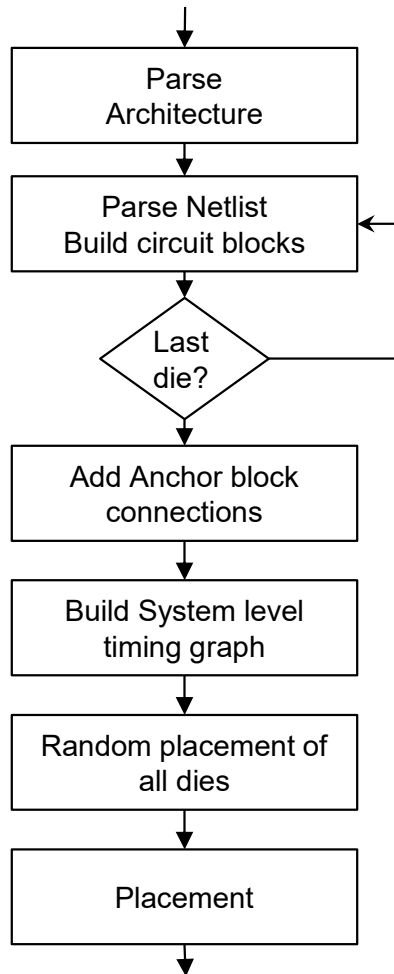
Partitioning

- Choosing the optimal value of UB is a tradeoff between
 - Minimizing the cutsize and obtaining balanced partitions



- Lower SLL utilization.
- Larger die size.
- Poor die resource utilization.

- Better die resource utilization.
- Smaller die size
- Slightly higher SLL utilization.



- SLL legalization is simpler.
- Positions fixed at the start of the flow -> parallel independent placement

Interconnection demand

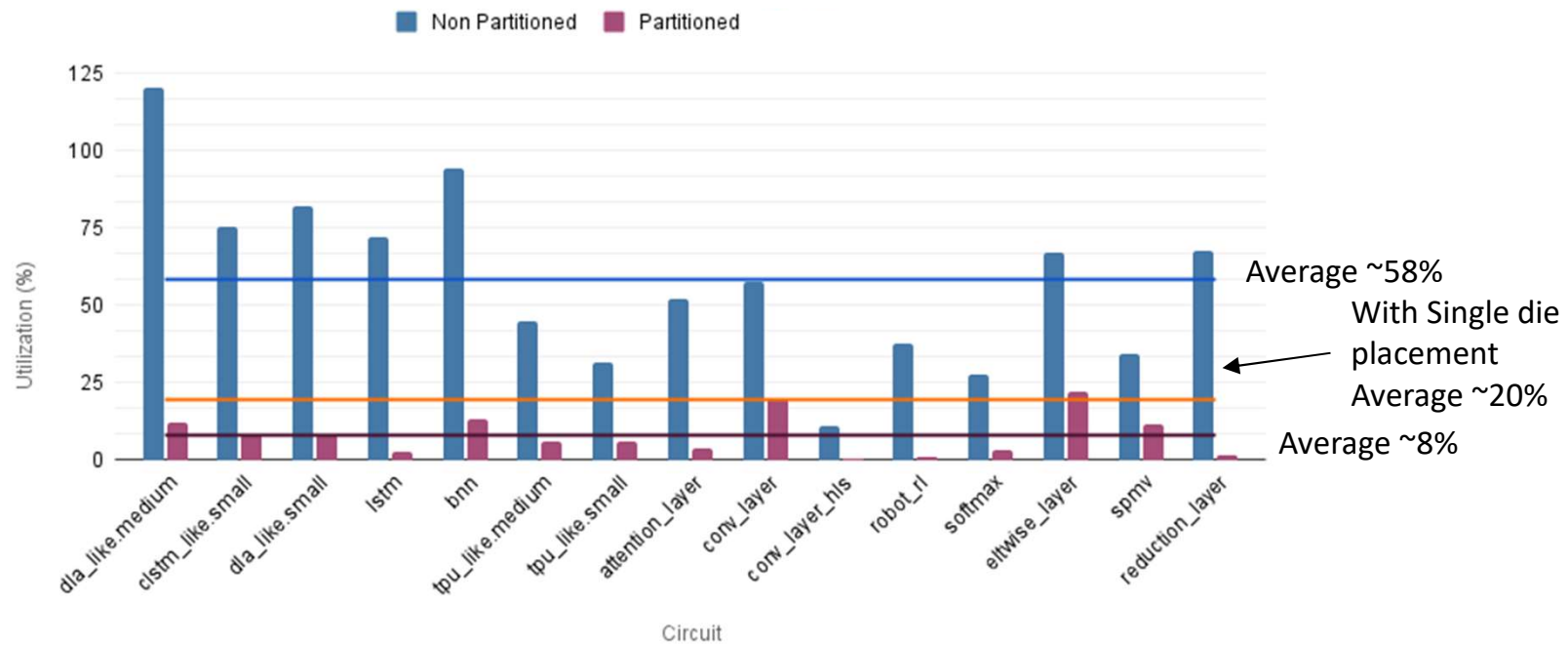


- From placement results
 - Get block positions
 - Get nets crossing the virtual boundary.
 - Nets crossing virtual boundary = Nets occupying an SLL.
- $\%Utilisation = \frac{Occupied\ SLLs}{Available\ SLLs}$

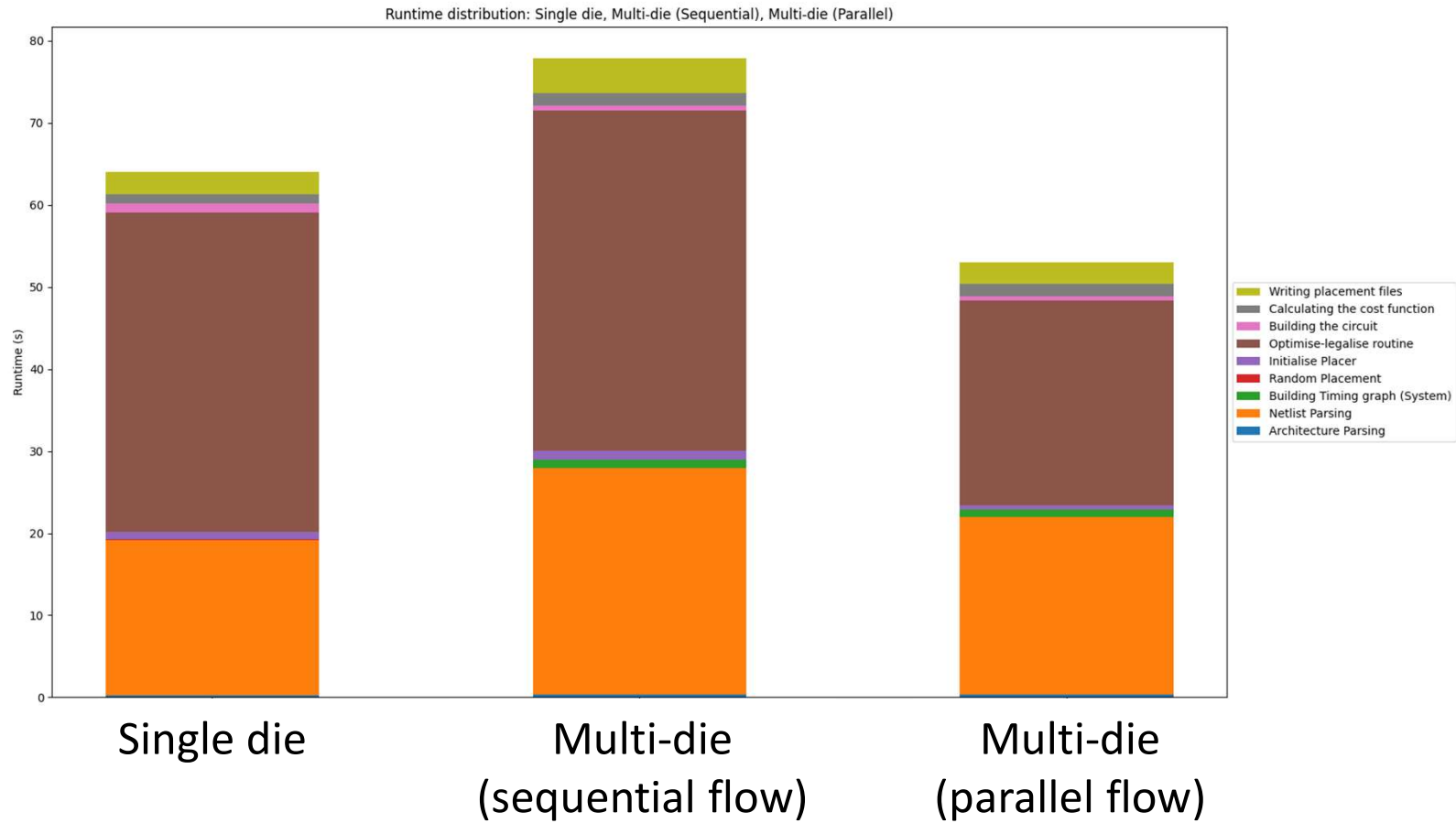
Interconnection demand

Multi-Die vs Single-die Placement

Interconnection Demand



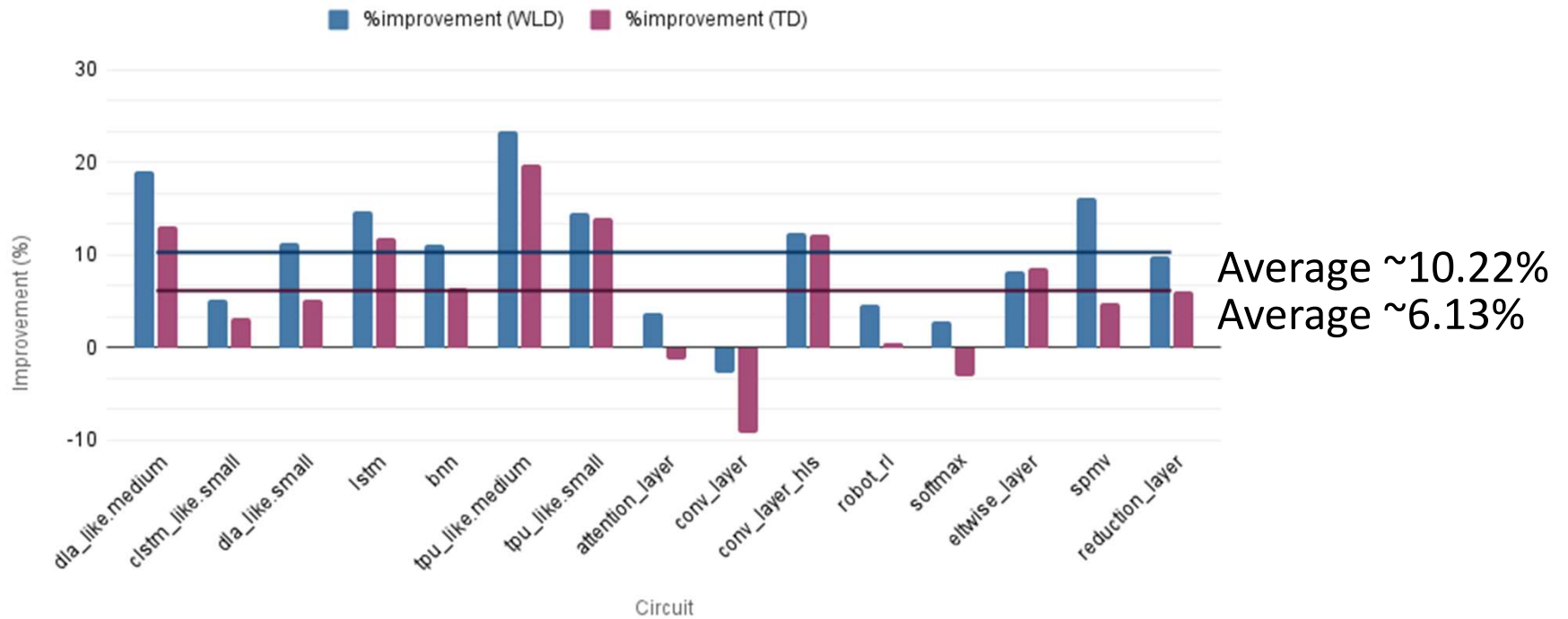
Runtime



WL estimation

Multi-Die vs Single-die Placement

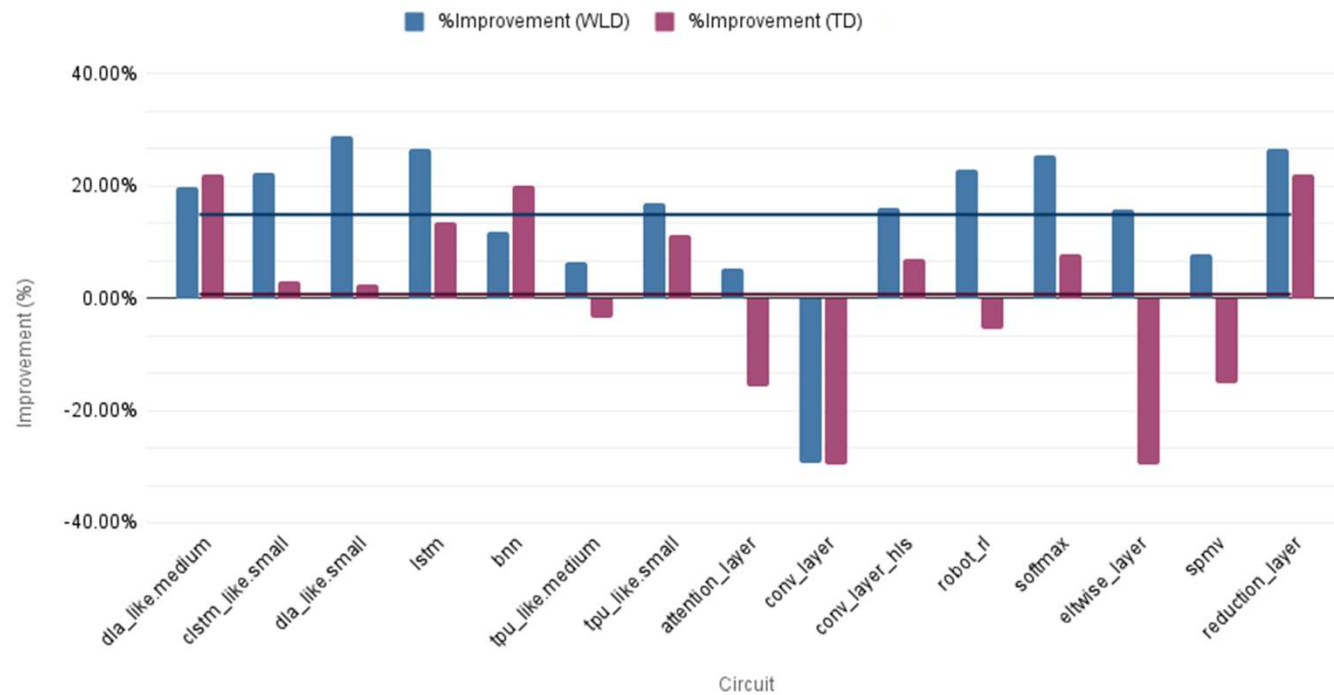
Estimated TWL



CPD estimation

Multi-Die vs Single-Die Placement

Estimated CPD



Average ~14%

Average ~0.65%

CRouteMD

- We are currently working on extending CRoute to a multi-die parallel router
- Unfortunately no results yet

Many thanks to my Ph.D. students who
collaborated to obtain these results:

Elias Vansteenkiste, Dries Vercruyce,

Yun Zhou, Raveena Raikar

Q & A