北京大學
PEKING UNIVERSITY

IEEE/ACM ICCAD
2024 INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN
43rd Edition

DeePoly

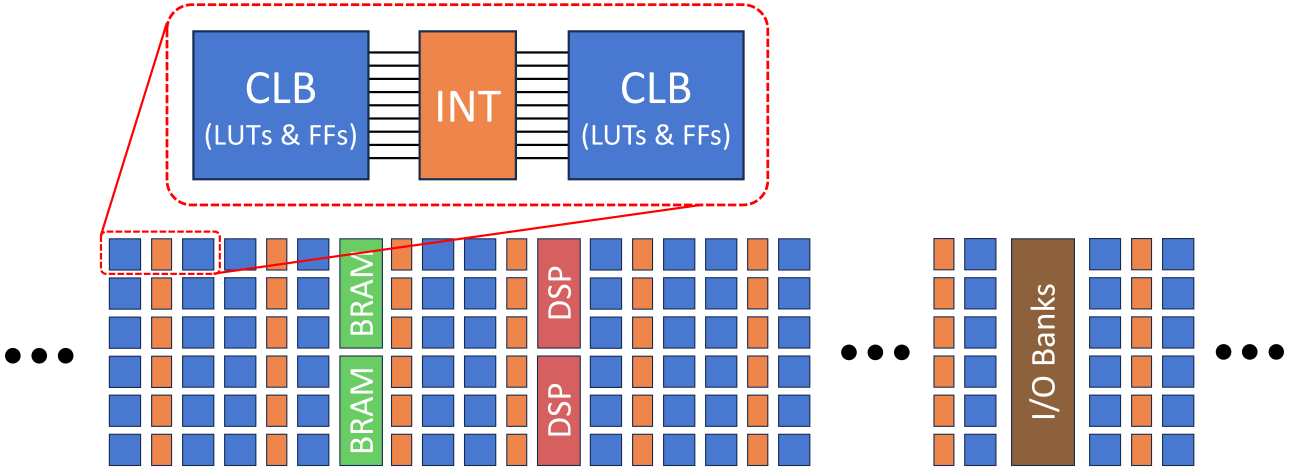
AceRoute: Adaptive Compute-Efficient FPGA Routing with Pluggable Intra-Connection Bidirectional Exploration

Xinming Wei[1], Ziyun Zhang[1], Sunan Zou[1], Kaiwen Sun[2], Jiahao Zhang[1], Jiaxi Zhang[1], Ping Fan[2], Guojie Luo[1]

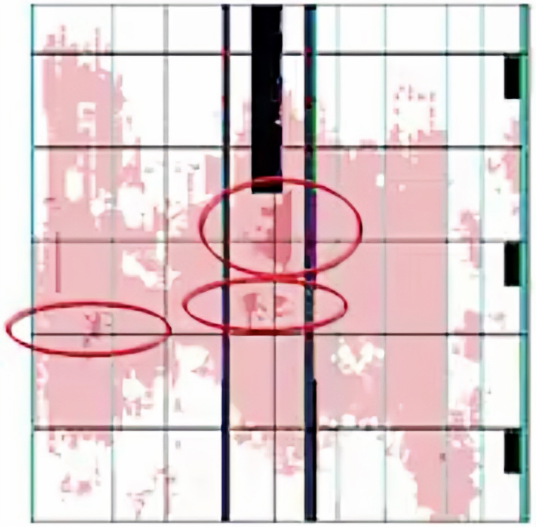
[1]Peking University [2]DeePoly Technology Inc.

FPGA Routing: Challenges

- More complex 1) FPGA architectures, and 2) circuit designs
- Vast congestions prevent routing closure



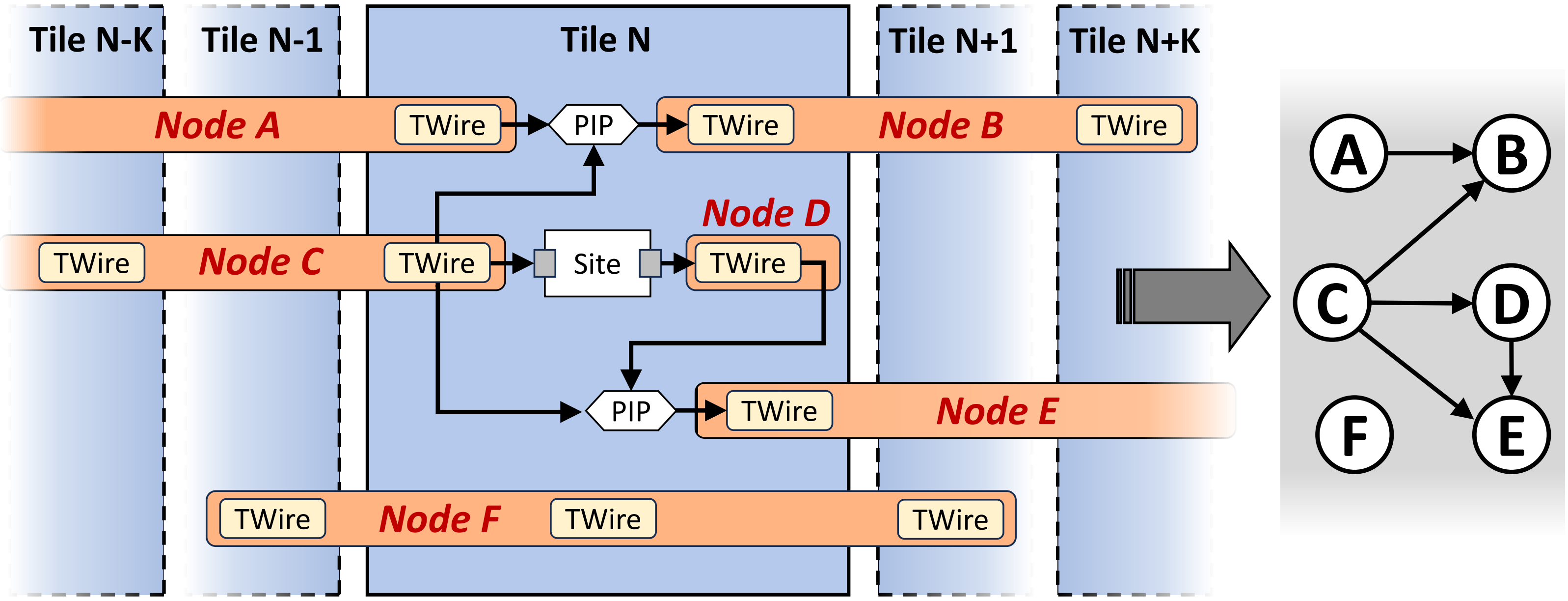
UltraScale+ architecture



Routing congestions (hotspots)

FPGA Routing: Formulation

- Disjoint-path problem on the RRG; NP-complete
    - Minimize wire length or delay



Routing Resource Graph (RRG)

Inter- vs. Intra- Connection Routing

Negotiated Congestion Routing *[Pathfinder, 1995]*

```
for (each routing iteration) {
   if (no overlap exists)  break;
   for (each congested connection) {
      Rip-up this connection;
      Route this connection with A* search;
      Save the result path and update
        the present cost of alongside nodes;
   }
   Update the history cost of all nodes;
}
```

Inter- vs. **Intra-** Connection Routing

Inter-Connection Speedup

- Connection routing order
- Recursive-partition parallel routing
- Overlap-tolerant parallel routing

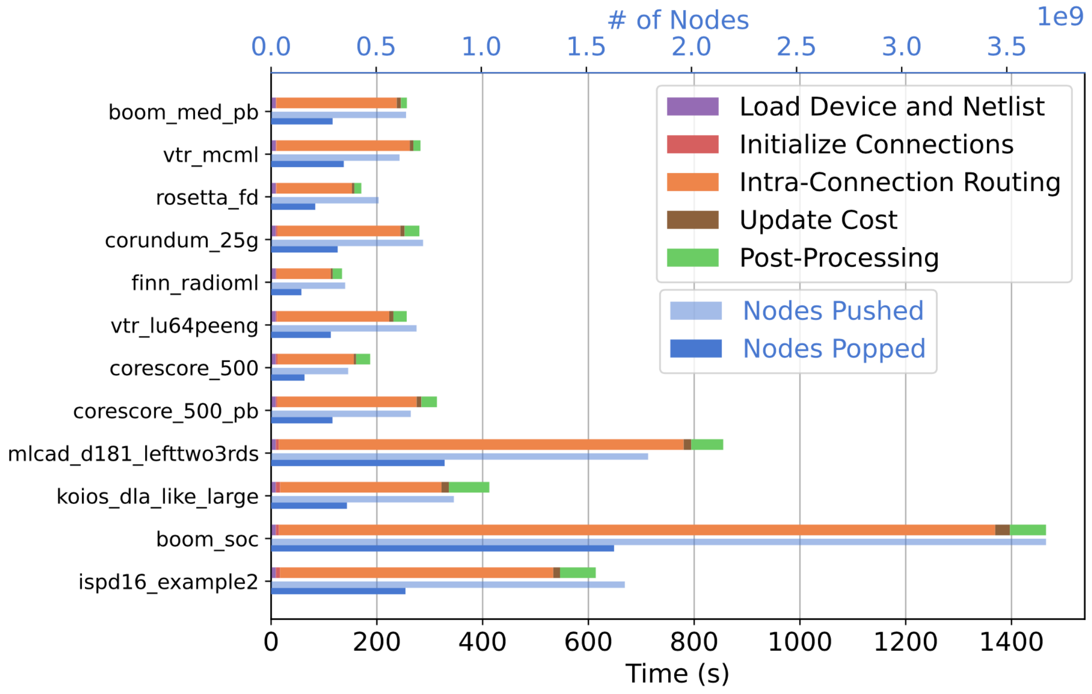
×

Intra-Connection Speedup

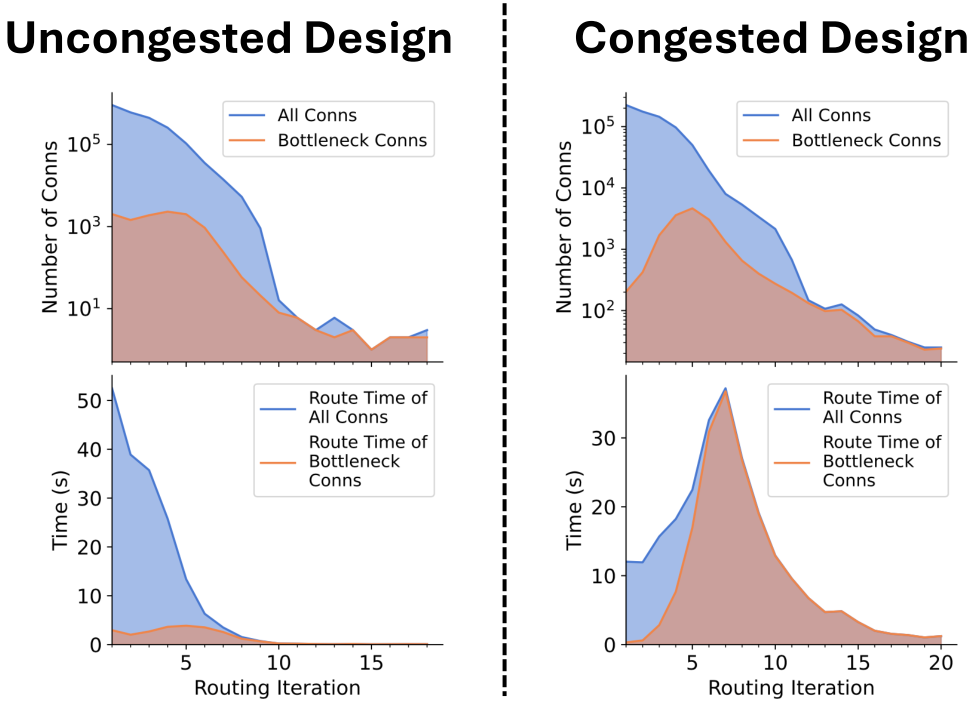
- Heuristic optimization of cost settings
- *Adaptive bidirectional search (this work!)*

Runtime Profiling

- Typical sequential router w/ *unidirectional A* exploration (UE)*
- <u>Bottleneck</u> connections in <u>congested</u> designs dominate runtime



Routing runtime breakdown


Uncongested Design

Congested Design

Impact of bottleneck connections

Bidirectional Exploration (BE)

UE: Forward Search Process

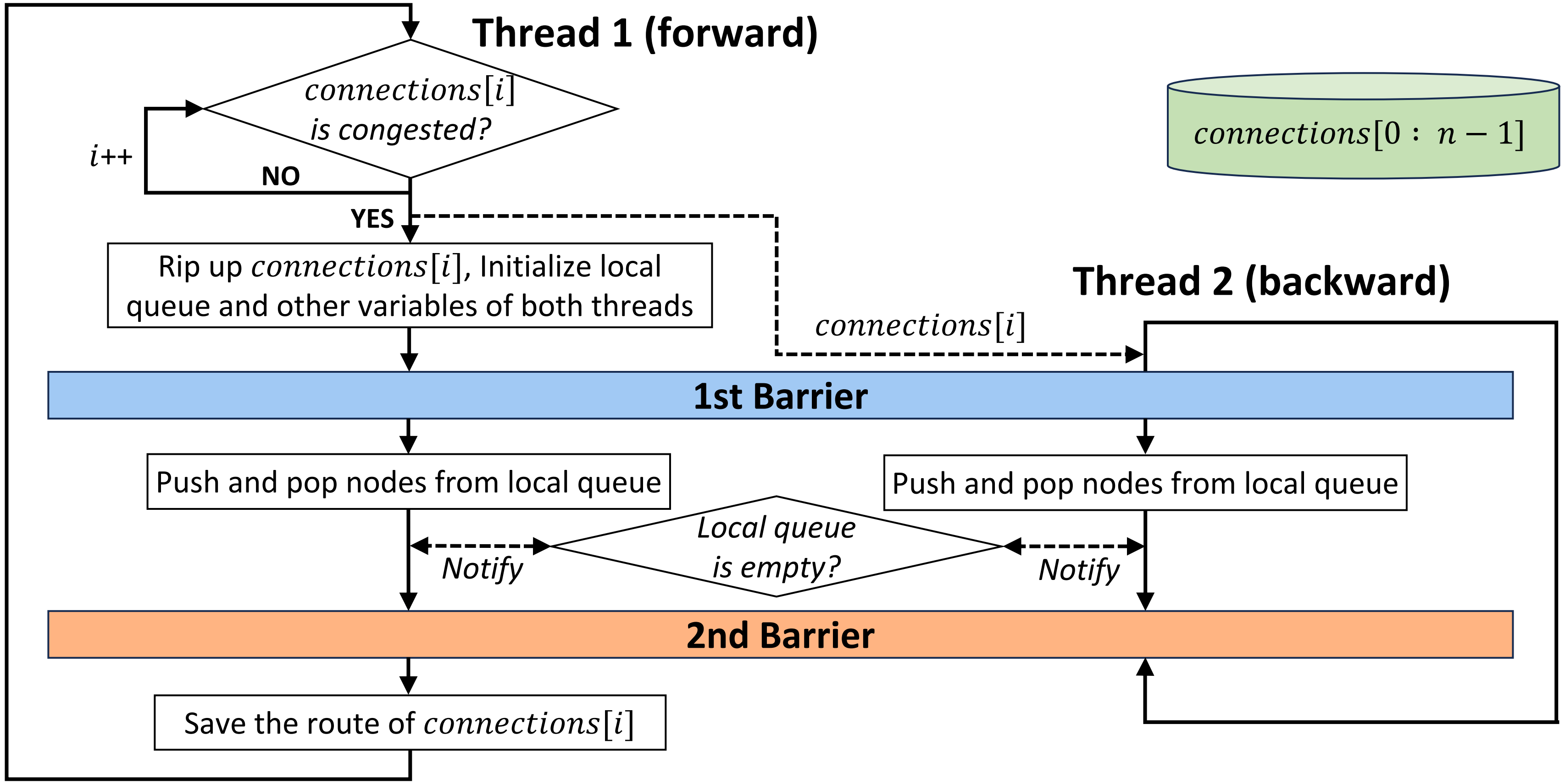
```
Init. priority queue PQ with source node;
while (PQ is not empty) {
  Pop the lowest-cost node u from PQ;
  for (each fanout v of node u) {
    Push v to PQ with Cost(u) + PathCost(v);
    if (v is sink) {
      return;
    }
  }
}
```

BE: Forward/Backward Search Process

```
Init. priority queue PQ with source node;
Init. shared best path cost L;
Init. lowest cost C* of this search frontier;
while (PQ is not empty) {
  Pop the lowest-cost node u from PQ;
  if (expand u is not promising)  continue;
  for (each fanout v of node u) {
    Push v to PQ with Cost(u) + PathCost(v);
    if (v is visited by the opposite proc) {
      if (this full path has cost < L)
        Update L and the result node with v;
    }
  }
  Update C* with the top of PQ;
}
```

BE expands much fewer resource nodes than UE when congested.

Bidirectional Exploration (BE)



Thread 1 (forward)

$connections[i]$ *is congested?*

i++

NO

YES

Rip up $connections[i]$, Initialize local queue and other variables of both threads

$connections[i]$

Thread 2 (backward)

$connections[0 : n-1]$

1st Barrier

Push and pop nodes from local queue

Push and pop nodes from local queue

Local queue is empty?

Notify

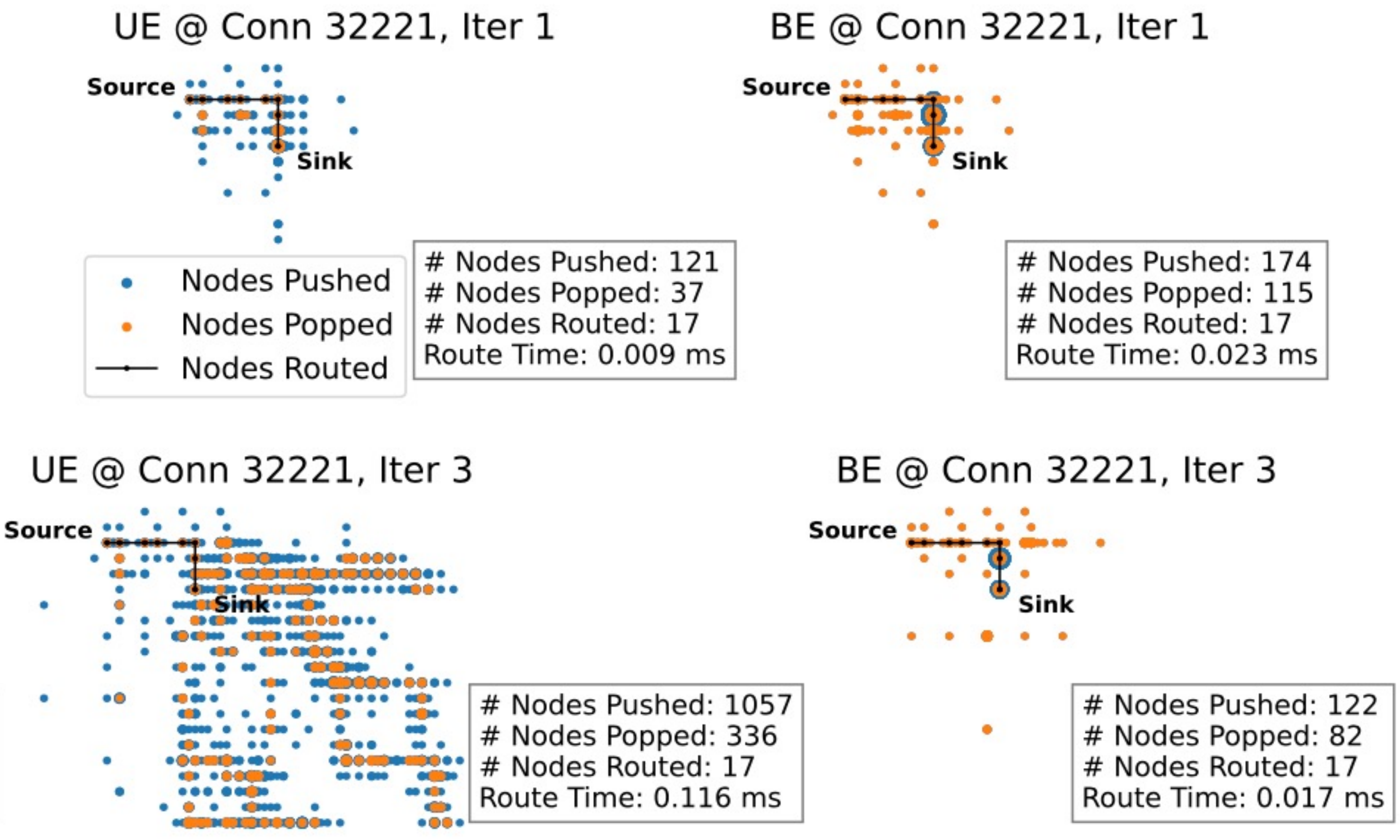
Notify

2nd Barrier

Save the route of $connections[i]$

Parallel (2-thread) BE

But Wait…Is BE All You Need?


UE @ Conn 32221, Iter 1

Nodes Pushed: 121
Nodes Popped: 37
Nodes Routed: 17
Route Time: 0.009 ms

BE @ Conn 32221, Iter 1

Nodes Pushed: 174
Nodes Popped: 115
Nodes Routed: 17
Route Time: 0.023 ms

UE @ Conn 32221, Iter 3

Nodes Pushed: 1057
Nodes Popped: 336
Nodes Routed: 17
Route Time: 0.116 ms

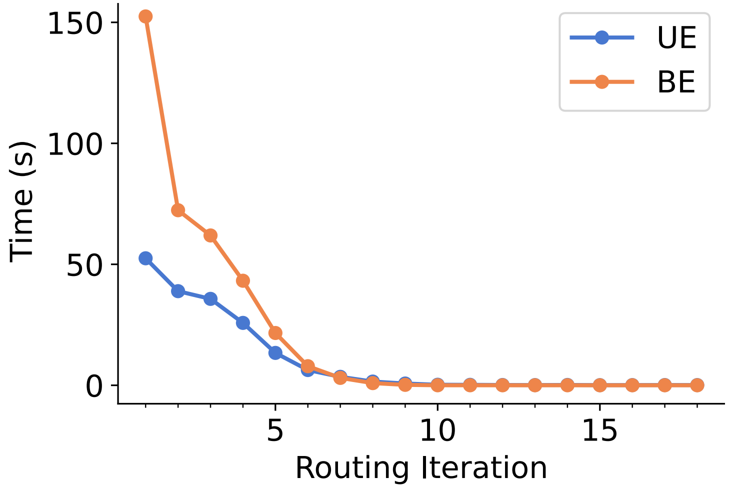
BE @ Conn 32221, Iter 3

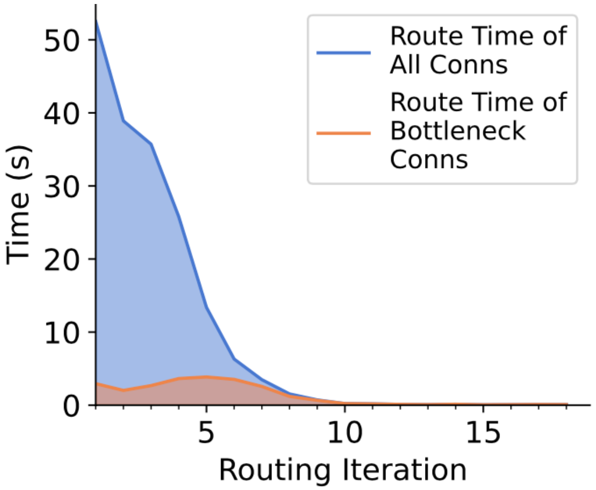
Nodes Pushed: 122
Nodes Popped: 82
Nodes Routed: 17
Route Time: 0.017 ms

Case study: route a specific connection with UE/BE at different routing stages

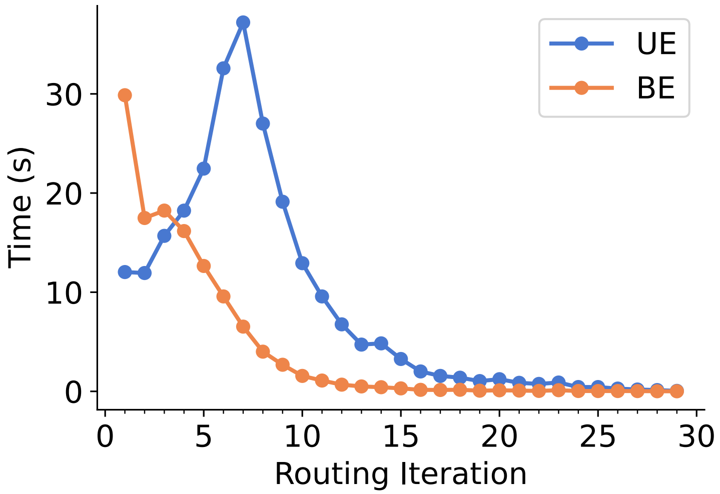
But Wait…Is BE All You Need?

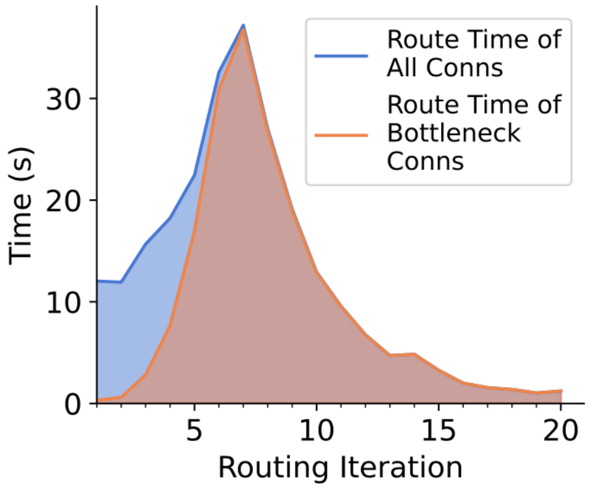
Uncongested Design





Congested Design

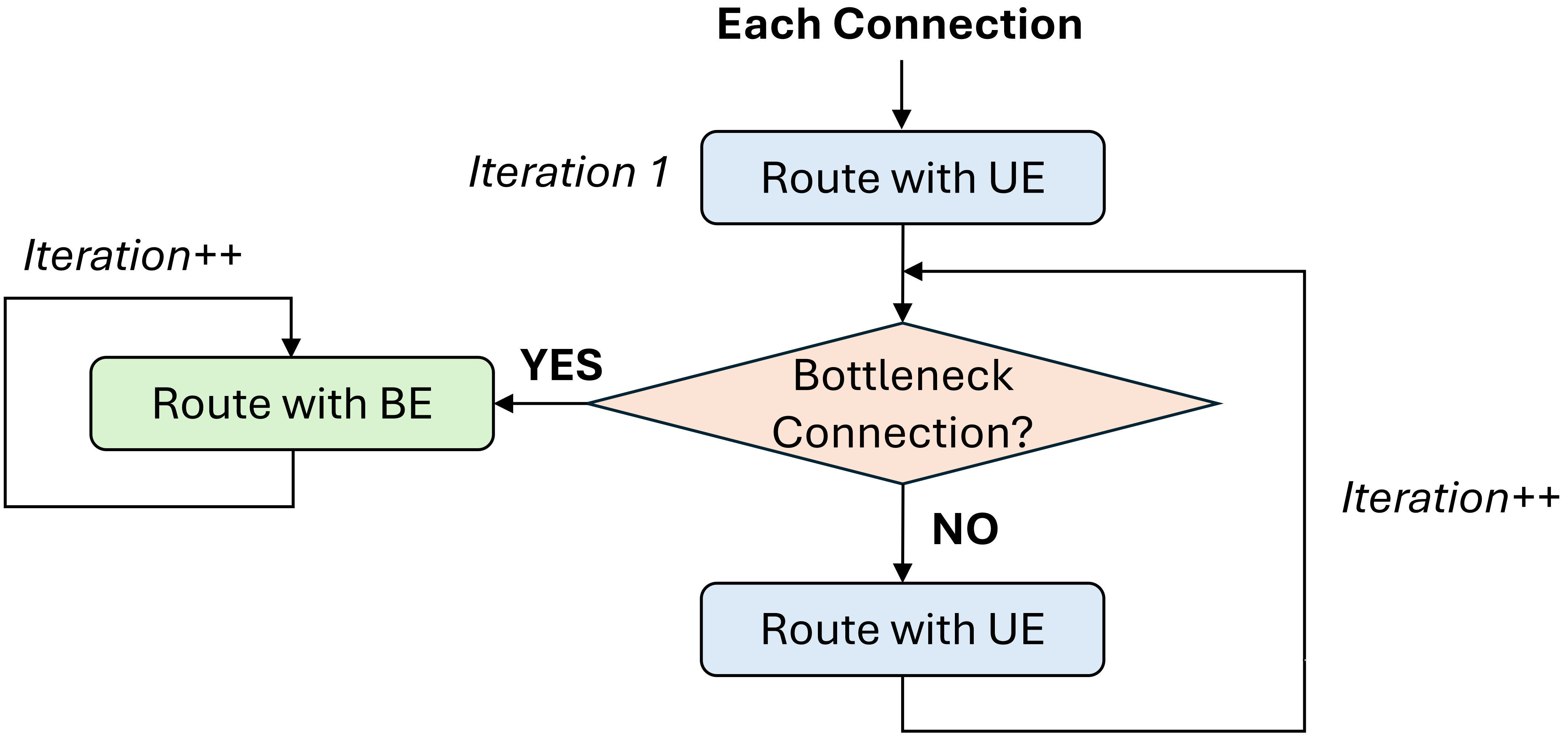




Compare UE and BE under different levels of congestion

Review the impact of bottleneck connections (p. 6)

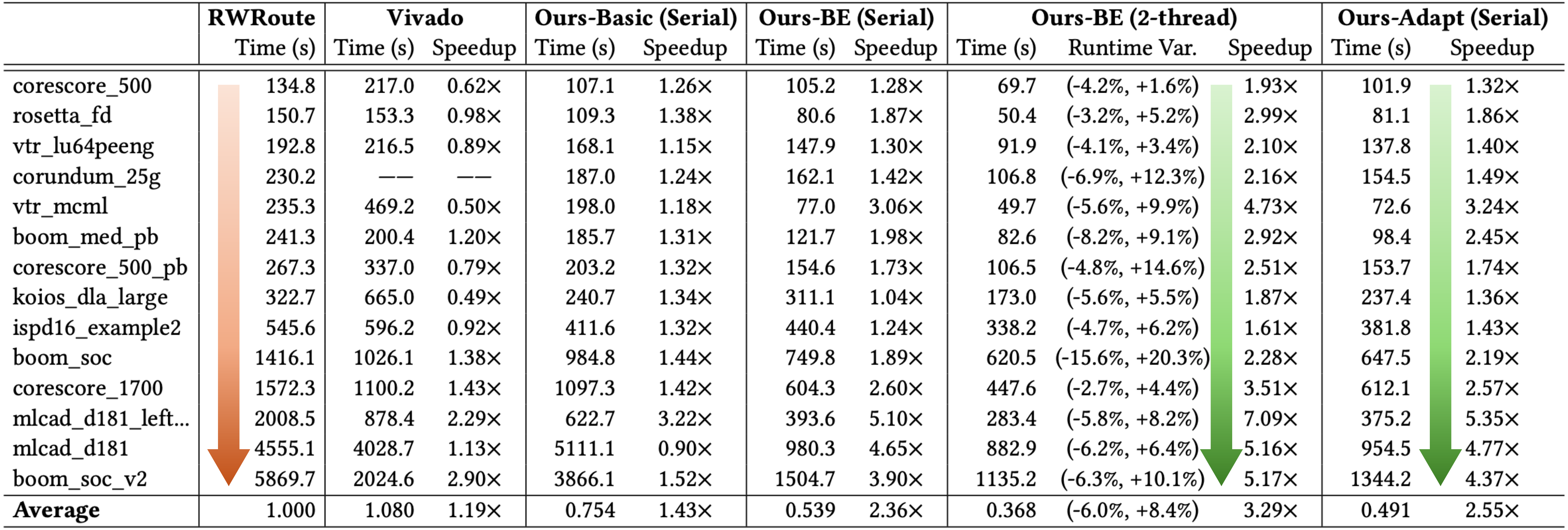
Adaptive BE


Each Connection
Iteration 1
Route with UE
Iteration++
Route with BE
YES
Bottleneck Connection?
NO
Route with UE
Iteration++

Setup and Implementation Details

- Our implementation
  - C++, ~6000 LOC
  - 4 versions: UE (serial), BE (serial), BE (2-thread), adaptive BE (serial)
- References
  - RapidWright RWRoute: a Java-based open-source FPGA router
  - Classical bidirectional A* algorithms
- Benmarks
  - FPGA’24 routing contest benchmarks

Intra-Connection Routing Results

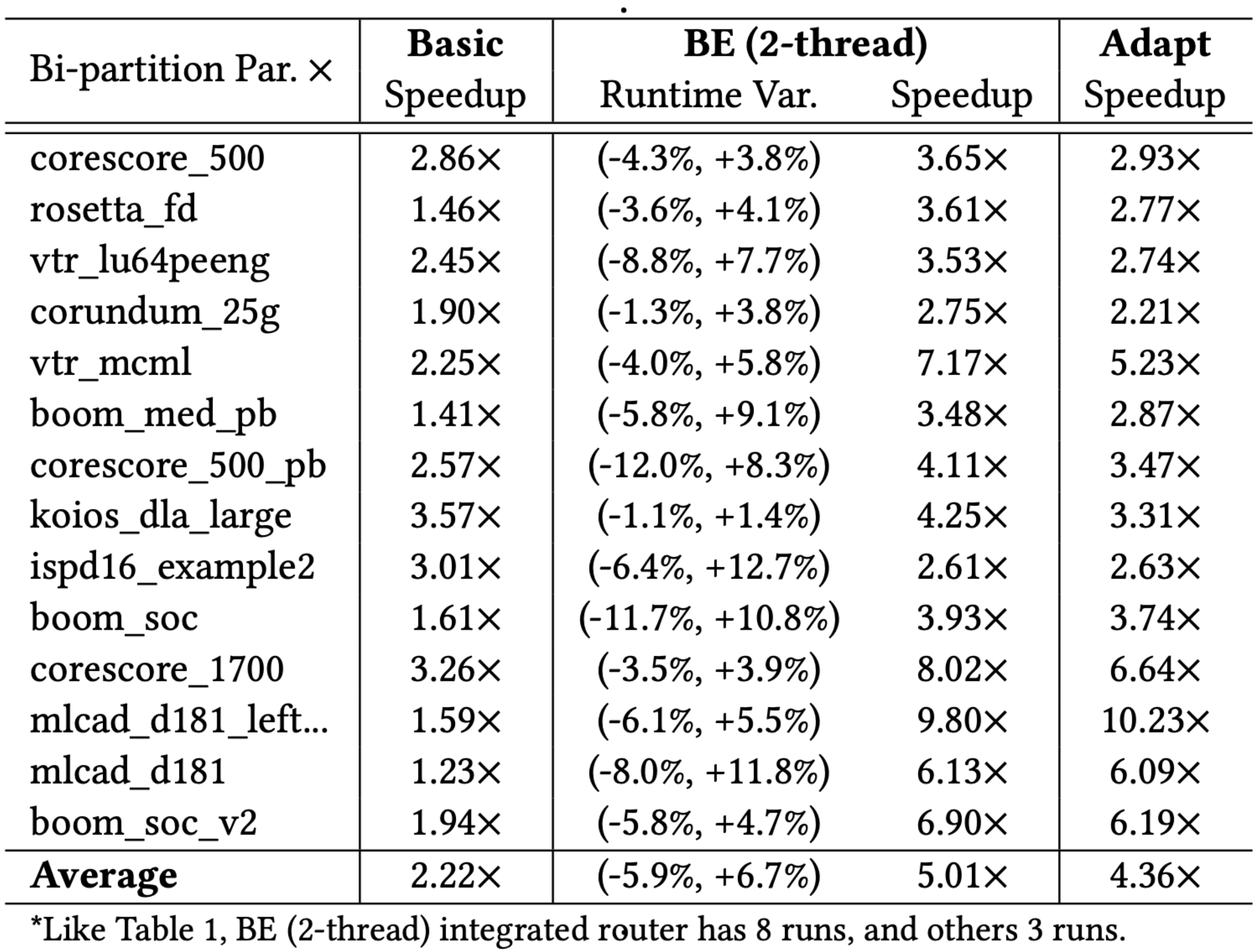


	RWRoute	Vivado		Ours-Basic (Serial)		Ours-BE (Serial)		Ours-BE (2-thread)			Ours-Adapt (Serial)	
	Time (s)	Time (s)	Speedup	Time (s)	Speedup	Time (s)	Speedup	Time (s)	Runtime Var.	Speedup	Time (s)	Speedup
corescore_500	134.8	217.0	0.62×	107.1	1.26×	105.2	1.28×	69.7	(-4.2%, +1.6%)	1.93×	101.9	1.32×
rosetta_fd	150.7	153.3	0.98×	109.3	1.38×	80.6	1.87×	50.4	(-3.2%, +5.2%)	2.99×	81.1	1.86×
vtr_lu64peeng	192.8	216.5	0.89×	168.1	1.15×	147.9	1.30×	91.9	(-4.1%, +3.4%)	2.10×	137.8	1.40×
corundum_25g	230.2	——	——	187.0	1.24×	162.1	1.42×	106.8	(-6.9%, +12.3%)	2.16×	154.5	1.49×
vtr_mcml	235.3	469.2	0.50×	198.0	1.18×	77.0	3.06×	49.7	(-5.6%, +9.9%)	4.73×	72.6	3.24×
boom_med_pb	241.3	200.4	1.20×	185.7	1.31×	121.7	1.98×	82.6	(-8.2%, +9.1%)	2.92×	98.4	2.45×
corescore_500_pb	267.3	337.0	0.79×	203.2	1.32×	154.6	1.73×	106.5	(-4.8%, +14.6%)	2.51×	153.7	1.74×
koios_dla_large	322.7	665.0	0.49×	240.7	1.34×	311.1	1.04×	173.0	(-5.6%, +5.5%)	1.87×	237.4	1.36×
ispd16_example2	545.6	596.2	0.92×	411.6	1.32×	440.4	1.24×	338.2	(-4.7%, +6.2%)	1.61×	381.8	1.43×
boom_soc	1416.1	1026.1	1.38×	984.8	1.44×	749.8	1.89×	620.5	(-15.6%, +20.3%)	2.28×	647.5	2.19×
corescore_1700	1572.3	1100.2	1.43×	1097.3	1.42×	604.3	2.60×	447.6	(-2.7%, +4.4%)	3.51×	612.1	2.57×
mlcad_d181_left...	2008.5	878.4	2.29×	622.7	3.22×	393.6	5.10×	283.4	(-5.8%, +8.2%)	7.09×	375.2	5.35×
mlcad_d181	4555.1	4028.7	1.13×	5111.1	0.90×	980.3	4.65×	882.9	(-6.2%, +6.4%)	5.16×	954.5	4.77×
boom_soc_v2	5869.7	2024.6	2.90×	3866.1	1.52×	1504.7	3.90×	1135.2	(-6.3%, +10.1%)	5.17×	1344.2	4.37×
Average	1.000	1.080	1.19×	0.754	1.43×	0.539	2.36×	0.368	(-6.0%, +8.4%)	3.29×	0.491	2.55×

- BE (2-thread): **3.2**× speedup (geomean) to RWRoute and Vivado.
- Adaptive BE (serial): **2.4**× speedup (geomean) to RWRoute and Vivado.
- Perform better on more complex/congested designs.

Enhanced Inter-Connection Parallelism

- Recursive-partition parallel inter-connection routing.
- With UE: **2.22×** speedup to RWRoute.
- With BE (2-thread): **5.01×** speedup to RWRoute.
- With adaptive BE (serial): **4.36×** speedup to RWRoute.



Bi-partition Par. ×	Basic	BE (2-thread)		Adapt
	Speedup	Runtime Var.	Speedup	Speedup
corescore_500	2.86×	(-4.3%, +3.8%)	3.65×	2.93×
rosetta_fd	1.46×	(-3.6%, +4.1%)	3.61×	2.77×
vtr_lu64peeng	2.45×	(-8.8%, +7.7%)	3.53×	2.74×
corundum_25g	1.90×	(-1.3%, +3.8%)	2.75×	2.21×
vtr_mcml	2.25×	(-4.0%, +5.8%)	7.17×	5.23×
boom_med_pb	1.41×	(-5.8%, +9.1%)	3.48×	2.87×
corescore_500_pb	2.57×	(-12.0%, +8.3%)	4.11×	3.47×
koios_dla_large	3.57×	(-1.1%, +1.4%)	4.25×	3.31×
ispd16_example2	3.01×	(-6.4%, +12.7%)	2.61×	2.63×
boom_soc	1.61×	(-11.7%, +10.8%)	3.93×	3.74×
corescore_1700	3.26×	(-3.5%, +3.9%)	8.02×	6.64×
mlcad_d181_left...	1.59×	(-6.1%, +5.5%)	9.80×	10.23×
mlcad_d181	1.23×	(-8.0%, +11.8%)	6.13×	6.09×
boom_soc_v2	1.94×	(-5.8%, +4.7%)	6.90×	6.19×
Average	2.22×	(-5.9%, +6.7%)	5.01×	4.36×

*Like Table 1, BE (2-thread) integrated router has 8 runs, and others 3 runs.

Conclusion

Efficient

Ultrafast routing for congested connections

Adaptive

Online UE/BE switching

AceRoute is…

Seamless integration into inter-connection parallel routers

Pluggable

Low coding efforts
Leverage existing routers

Light-weighted

Thank you!

Q & A

Contact:

Xinming Wei (weixinming@pku.edu.cn)
Jiaxi Zhang (zhangjiaxi@pku.edu.cn)
Guojie Luo (gluo@pku.edu.cn)