FPGA-Accelerated Maze Routing Kernel for VLSI Designs

Xun Jiang[1], Jiarui Wang[2], Yibo Lin[2], and Zhongfeng Wang[1]

[1] ICAIS Lab, School of Electronic Science and Engineering, Nanjing University

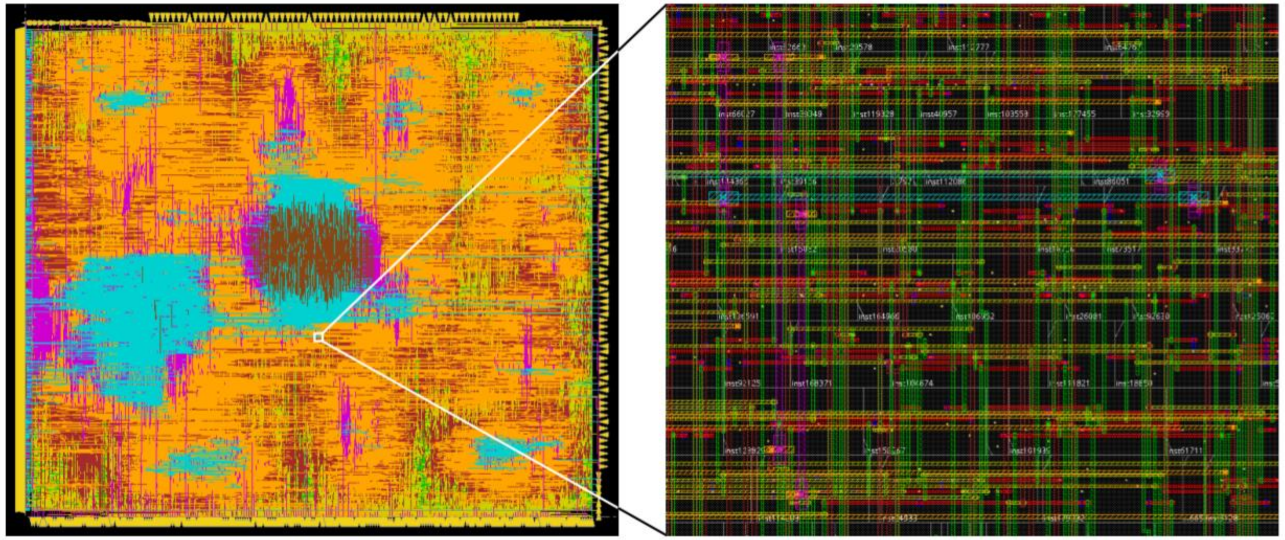
[2] CECA, CS Department, Peking University

ASP-DAC, Jan. 17-20, 2022, Virtual Conference

Outline

Outline

Detailed Routing

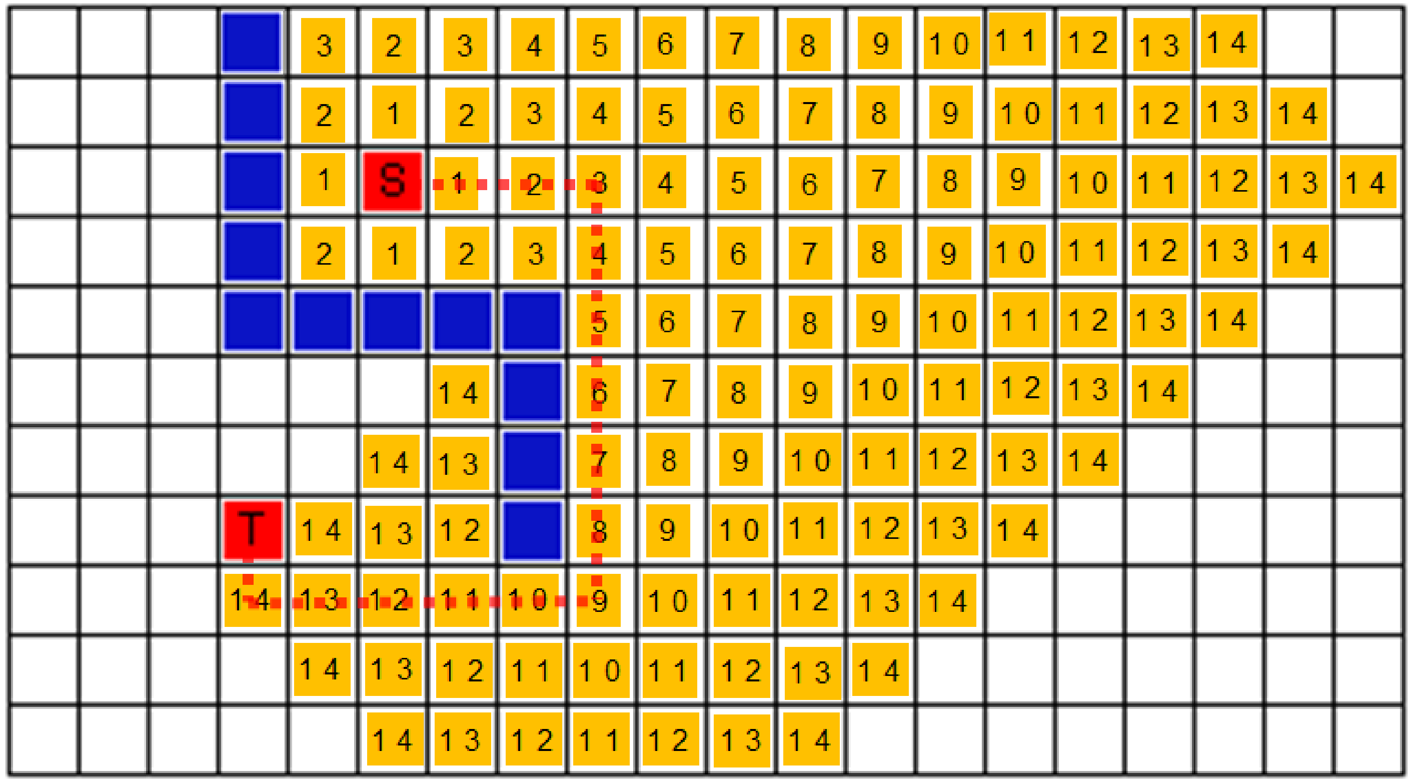


Solution of Dr.CU on ISPD18 test10 [1]

- Main Challenges
  - ◆ Complicated design rules
  - ◆ Large solution space ($10^4 \times 10^4 \times 10$ grid graph)
  - ◆ More time-consuming with new technology

[1] Dr. CU: Detailed routing by sparse grid graph and minimum-area-captured path search, TCAD, 2019

Maze Routing Kernel



- Searching Space:
  - ◆ 2D/3D Grid Graph
- Searching Process:
  - ◆ Select vertex with minimal cost
  - ◆ Expand frontier
  - ◆ Check terminal vertex
  - ◆ Reconstruct path

Image source: https://vlsi-soc.blogspot.com/2018/12/maze-router-lees-algorithm.html

FPGA Acceleration

Genomics
Astrophysics
Big Data Analysis
Cyber Security
Financial
Genomics
Machine Learning
Molecular Dynamics
Oil & Gas
Weather & Climate

Next is EDA?

Image source: https://www.xilinx.com/applications/data-center/high-performance-computing.html

Previous Works and Challenges

- Previous Works
  - ◆ Detailed routing: TritonRoute[ICCAD’18], Dr. CU[TCAD’20], and et al
    - ➢ Only leverage parallelization on CPU
  - ◆ FPGA-accelerated FPGA routing: [Korolija et al, IPDPSW’19]
    - ➢ 4-6x slower than Intel Core i5
- Challenges
  - ◆ Data dependency in maze routing
  - ◆ Different size of nets
  - ◆ A large number of random memory access

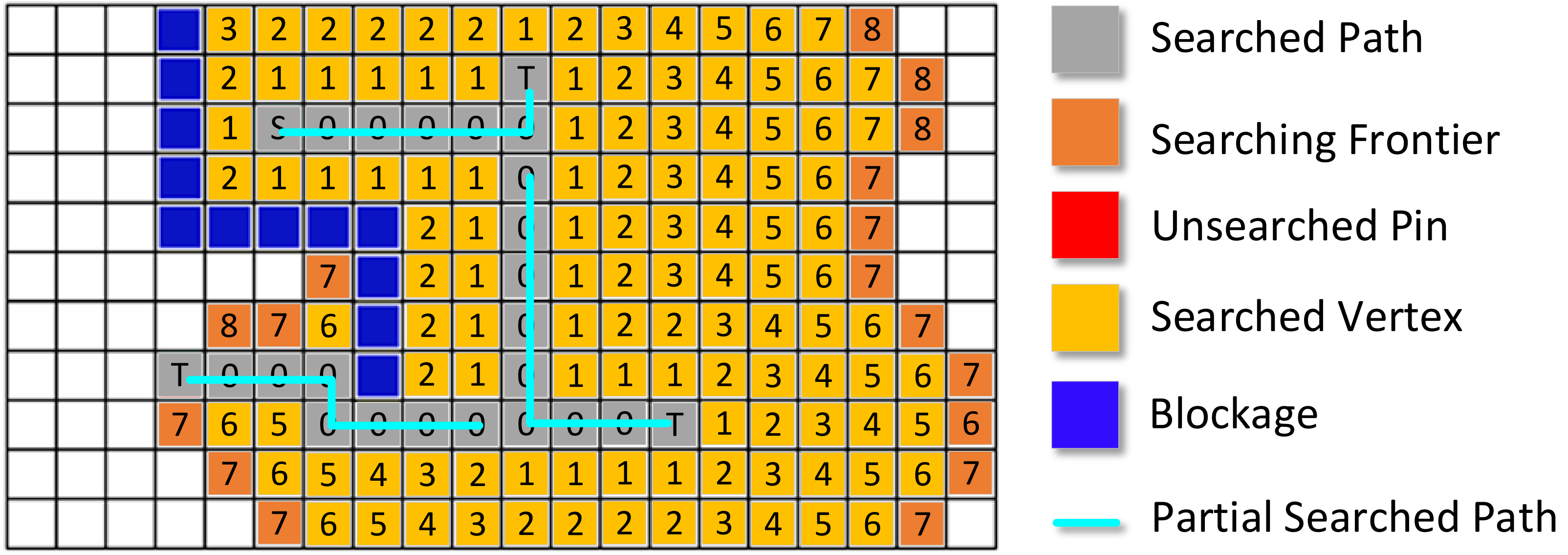
Outline

Multi-Pin Maze Routing

- **Goal:** Connect all pins on the net with minimal cost
- **Input:** A grid graph ***G(V, E)*** , ***N*** sets of vertices ***{S$_n$}*** related to the set of pins ***{p$_n$}***
- **Output:** Path ***P*** with minimum total cost

Cost Metrics:

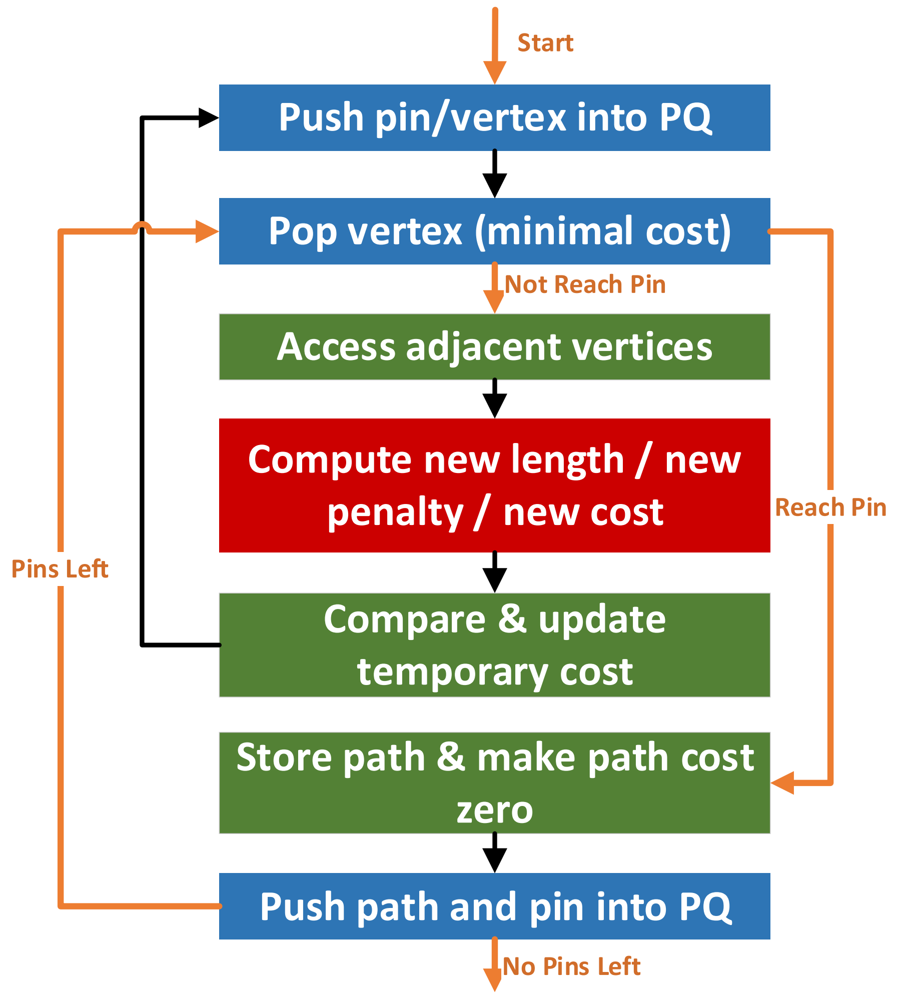
1. Total wire length
2. Total via count
3. Non-preferred usage
4. Design rule violations



Path ***P*** = Set of Partial Searched Paths

Hardware Optimization Methods

Baseline Software Dr.CU[2]


Start
Push pin/vertex into PQ
Pop vertex (minimal cost)
Not Reach Pin
Access adjacent vertices
Compute new length / new penalty / new cost
Reach Pin
Pins Left
Compare & update temporary cost
Store path & make path cost zero
Push path and pin into PQ
No Pins Left

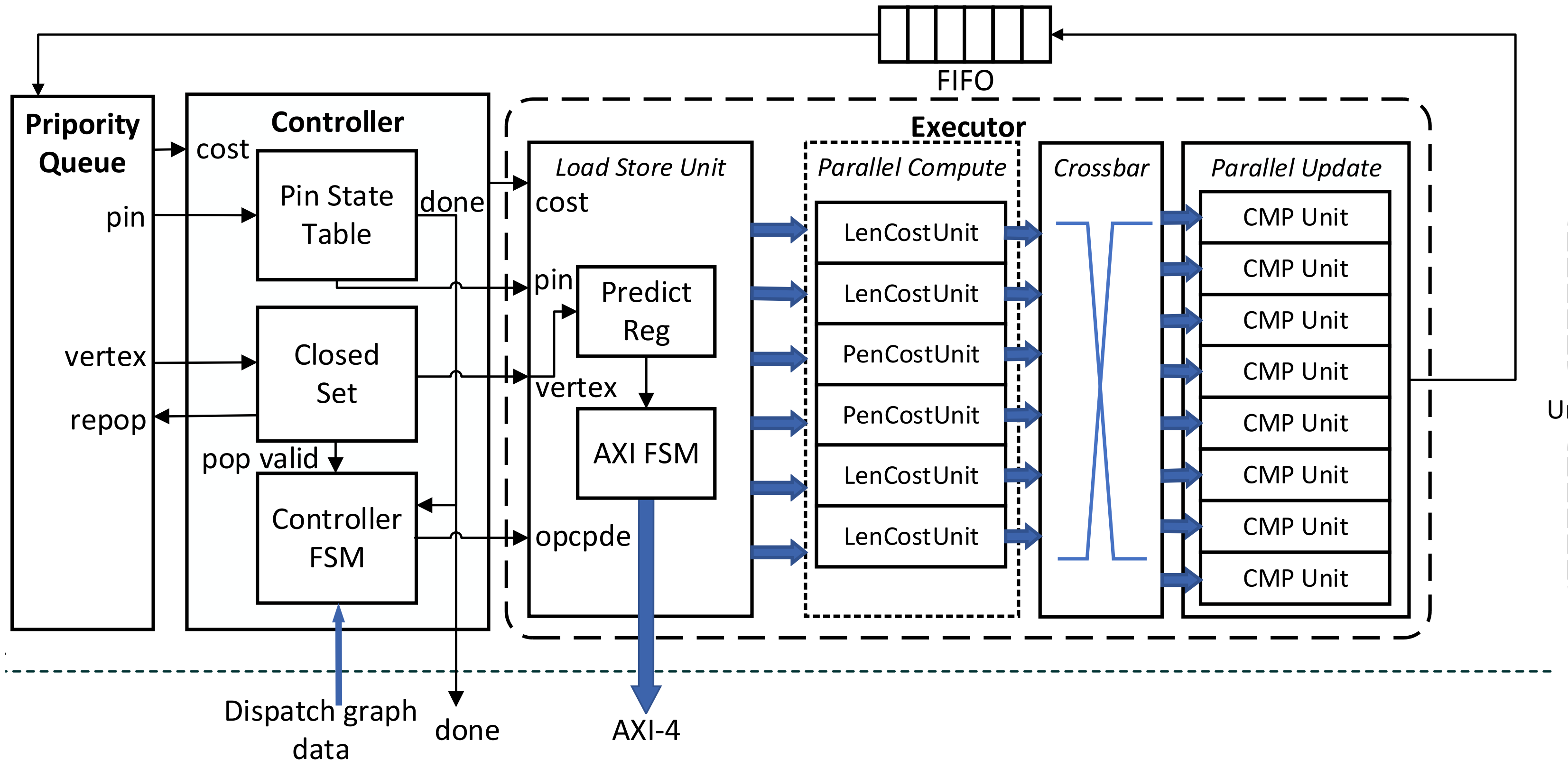
[2] https://github.com/cuhk-eda/dr-cu

Hardware Optimization Methods

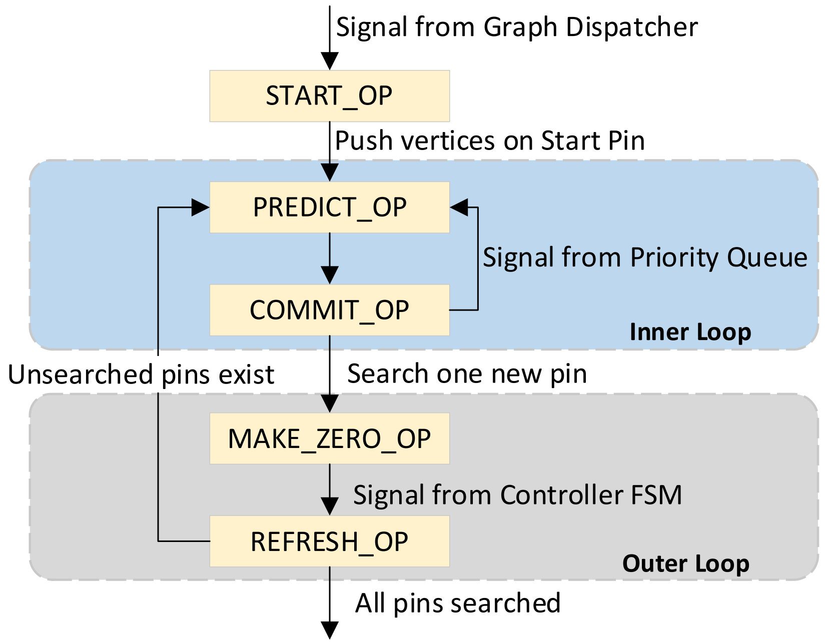
- Priority Queue Operation:
  - ◆ Pipelined hardware priority queue
  - ◆ Prediction of priority queue
- Computing Dominated Part:
  - ◆ Fast hardware circuit
- Memory Access Dominated Part:
  - ◆ Compact graph data structure
  - ◆ Temporary cost reuse on chip
- Complex Control Flow:
  - ◆ Predefined operation and schedule

Maze Routing Processing Element

Maze Routing PE Structure



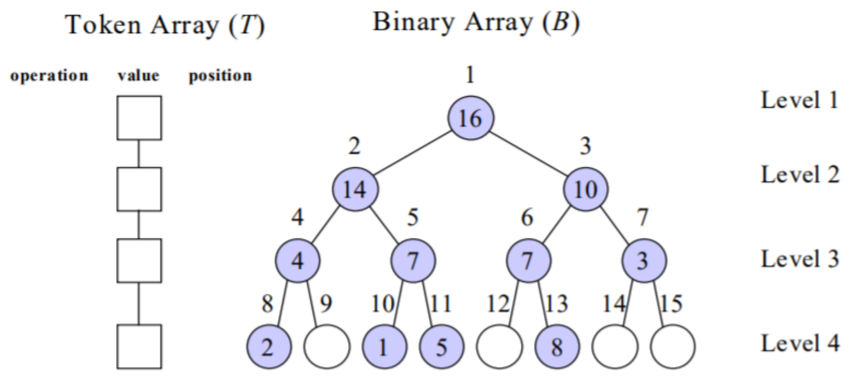
Operations and schedule



- Priority Queue : sort vertices by temporary cost in ascending order
- Controller: schedule different operations of hardware shown in the right picture
- Executor: compute the new length/penalty/cost of adjacent vertices and update the temporary cost

Tri-State Priority Queue

Base architecture: P-Heap [3]



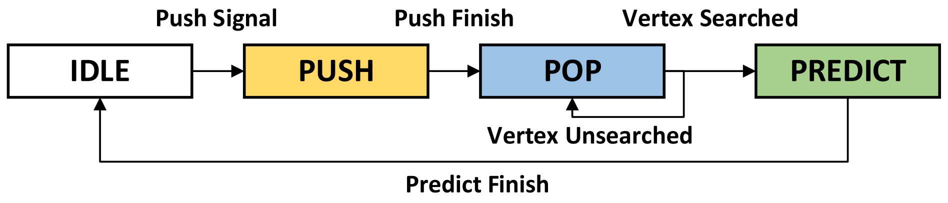
Time complexity: *O(1)*

FPGA LUT resource: *O(log(n))*

FPGA SRAM resource: *O(n)*

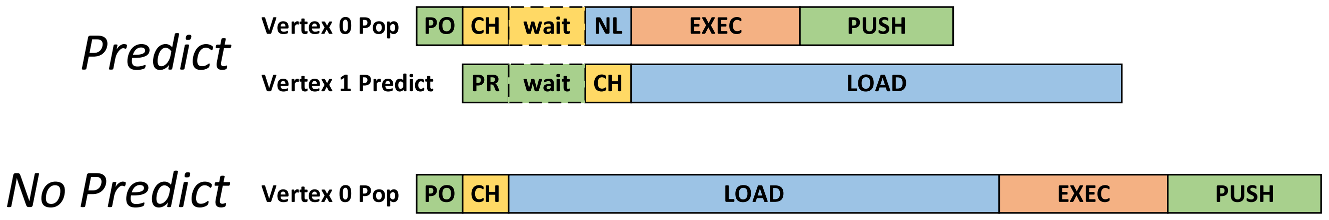
Only leverage internal parallelism of Priority Queue

Priority Queue with Prediction:



Predict Policy: Predict the new top vertex before the last popped vertex's adjacent vertices pushed into the Priority Queue.

Analysis of prediction in timeline:



Also leverage external parallelism of Priority Queue

Provide 7% performance increase on realistic FPGA environment

[3] R. Bhagwan and B. Lin, "Fast and scalable priority queue architecture for high-speed network switches," in Proceedings IEEE INFOCOM 2000

Graph Data Structure

(1) Graph Data

①	Graph Pin Table
②	Pin Vertex Table
③	Chip Data Block
④	Vertex Property Table

(2) Vertex Data

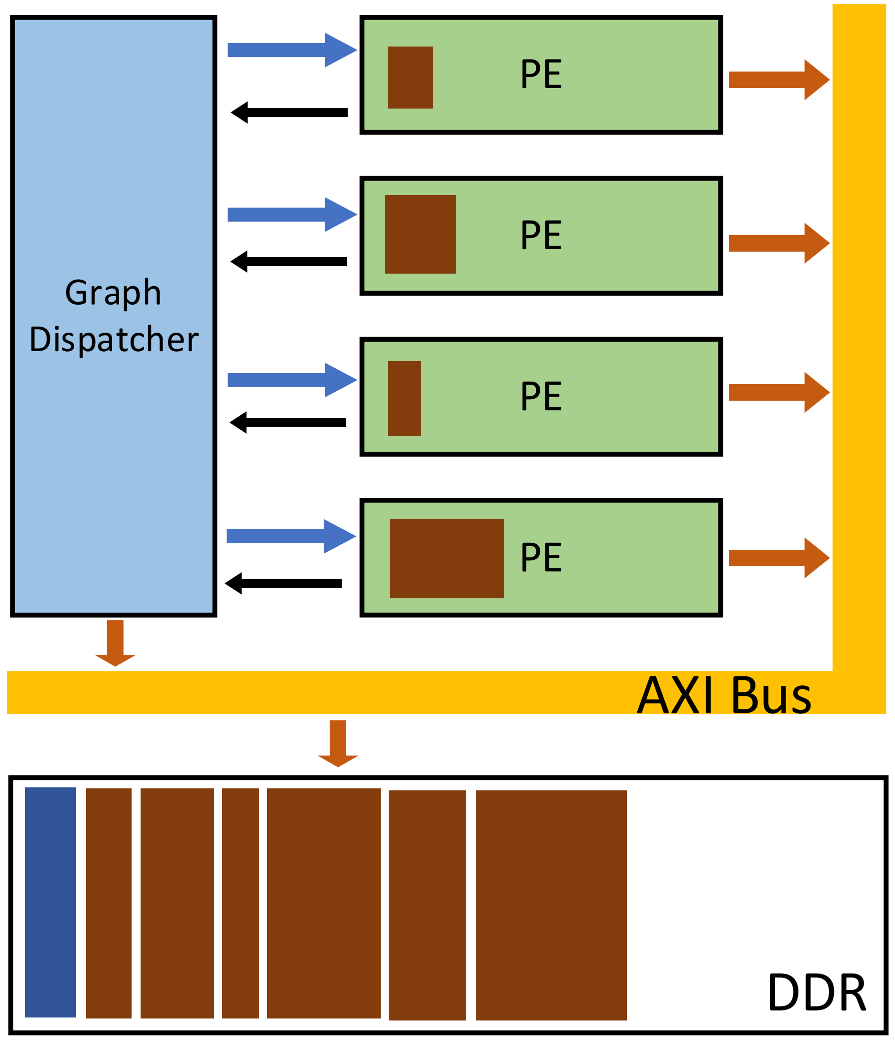
EdgeCost1								EdgeCost0			
EdgeCost3								EdgeCost2			
EdgeCost5								EdgeCost4			
newLenAdd1								newLenAdd0			
newLenAdd5								newLenAdd4			
Vertex3				Vertex2				Vertex1		Vertex0	
M	uL	L5	L4	L3	L2	L1	L0	Vertex5		Vertex4	
				Pin5		Pin4		Pin3	Pin2	Pin1	Pin0

Adjacent List Data Structure

Contiguous graph data ➔ Better data transfer via PCIe interface

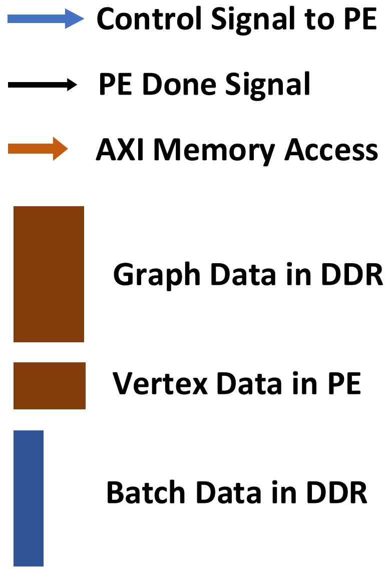
Compact vertex data ➔ Better memory access on DRAM

Batch Nets Parallelization



Functions of Graph Dispatcher:

1. Start Maze Routing PEs and monitor their state
2. Schedule batched graphs



Schedule Policy:

1. Fixed Priority
2. Start PE whenever PE is idle until no workloads reserved in DDR

Principle:

1. Make all PEs busy
2. PEs run in parallel independently

Outline

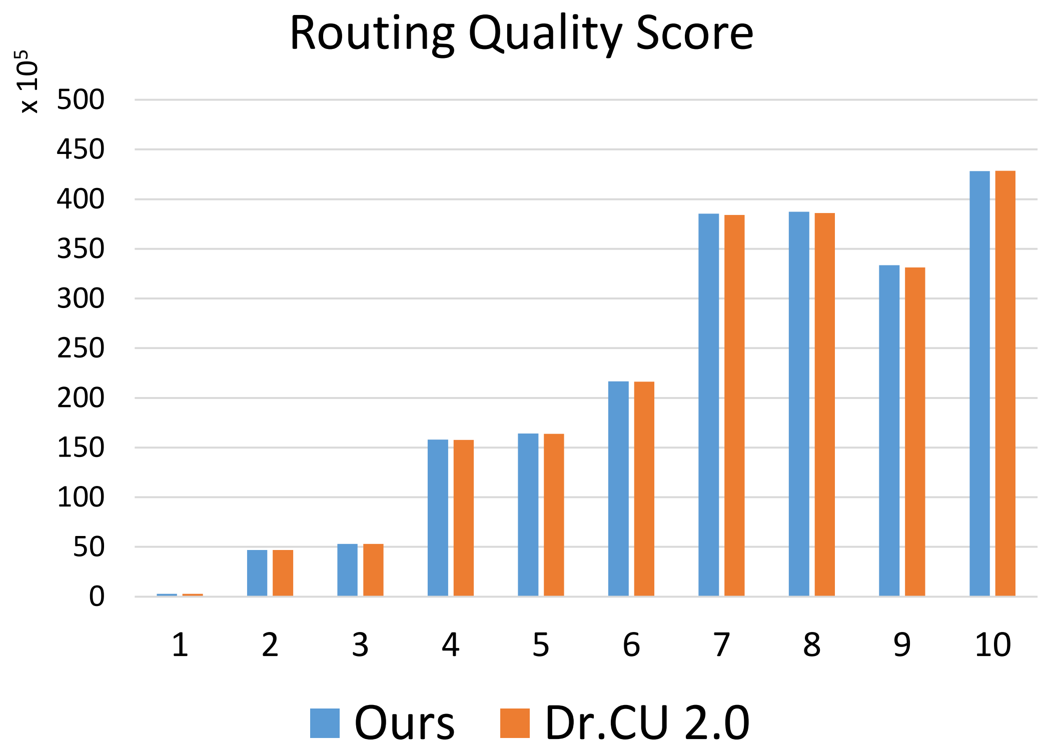
- Introduction
- Design Methodology
- **Experiment**
- Conclusion

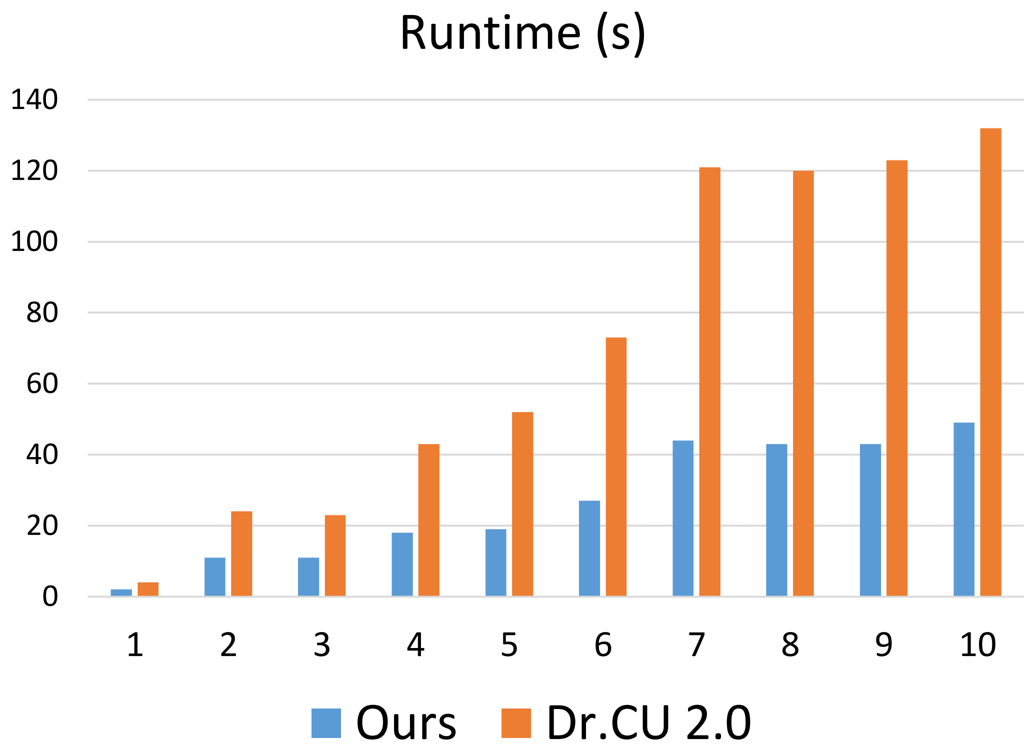
Experiment Settings

- Experimental Platform (Amazon EC2 f1.2xlarge instance)
  - ◆ 8-core Intel Xeon CPU E2-2686 v4 @2.30GHz, 122GB Memory
  - ◆ Xilinx Ultrascale+ VU9P FPGA, 1 16GB DDR4 Interface
- Configurations
  - ◆ CPU : 1/2/4/8 threads
  - ◆ FPGA : 1/2/4/8/12 PEs, 125MHz frequency
- Test sets
  - ◆ ISPD 2018 contest benchmarks[3]
  - ◆ Node number of net less than 2^{16} (more than 90% of all nets for large batch)

[3] S. Mantik et al "ISPD 2018 initial detailed routing contest and benchmarks," ISPD 2018

Routing Quality & Runtime

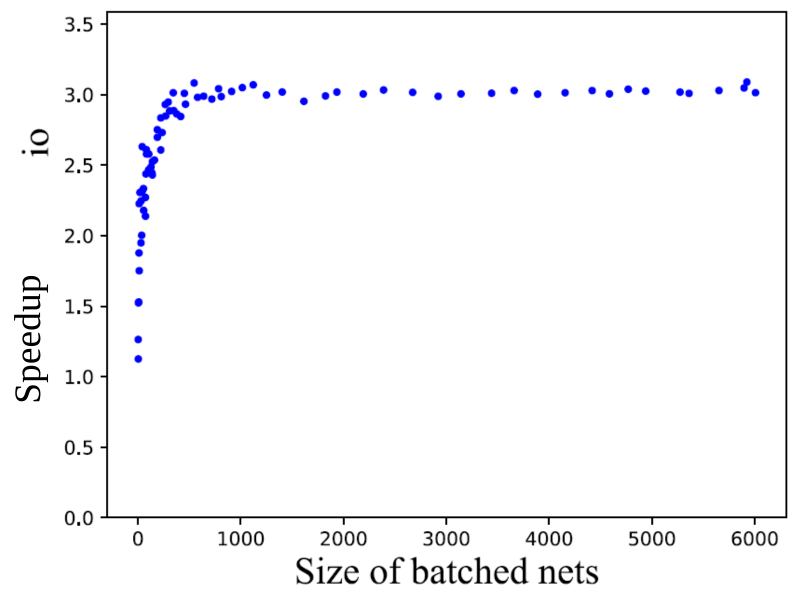




- CPU with 8 threads (maximum performance), FPGA with 12 PEs
- Runtime is the sum of the time of tested nets running in the first iteration
- Quality degradation is less than 1%
- The speed of each test is above 2x; the speedup of large tests is almost 3x

Speedup & Scalability

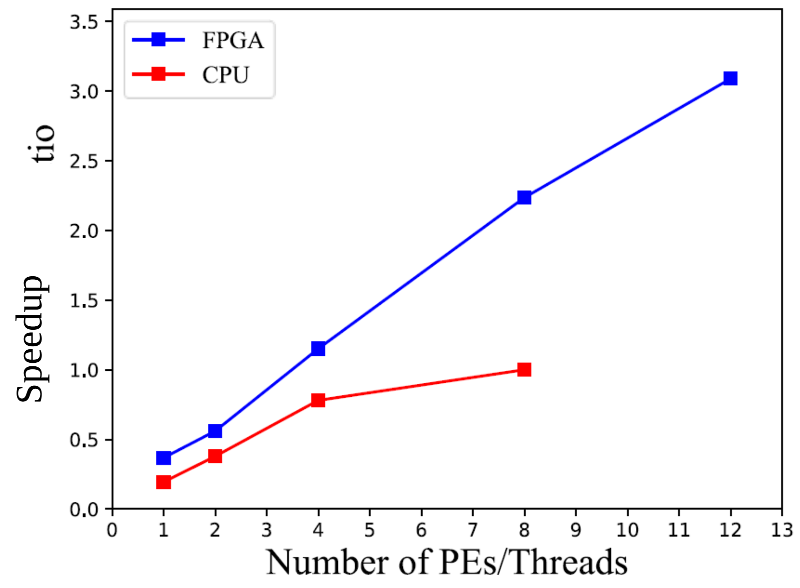
Speedup with **batch size**



$$\text{Speedup} = \frac{\#CPU\ runtime(8\ threads)}{\#FPGA\ runtime(12\ PEs)}$$

Batch nets are taken from ISPD2018 test 6

Speedup with **PEs/Threads number**

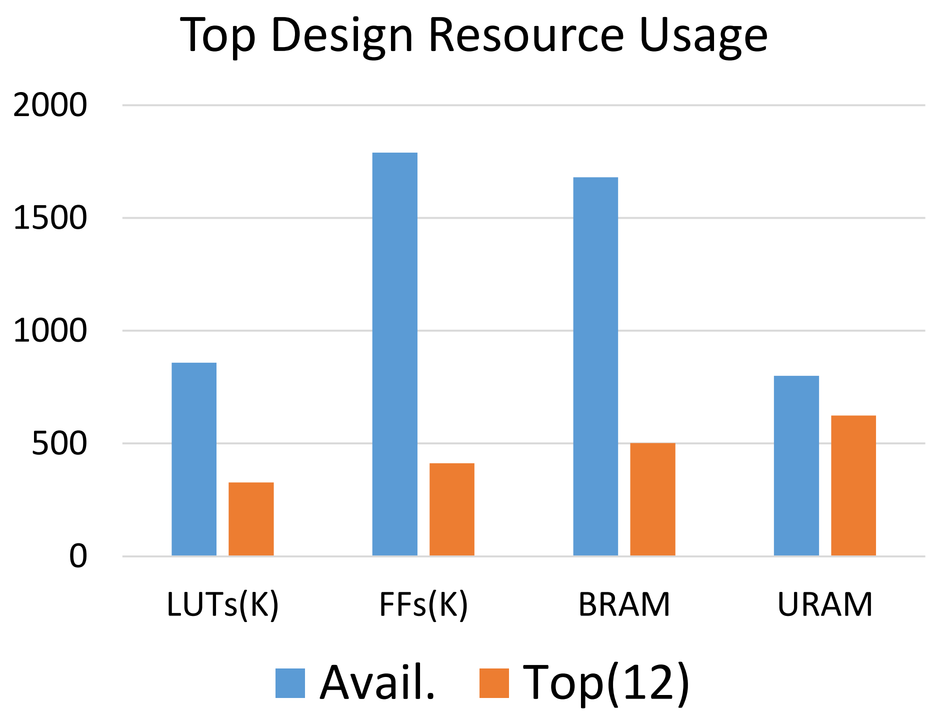


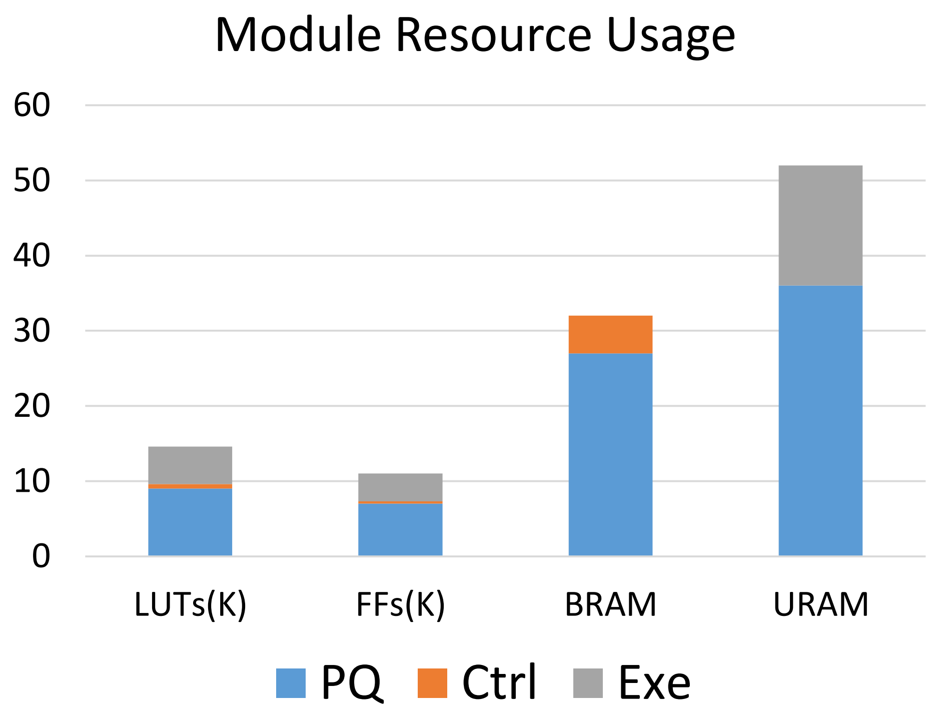
$$\text{Speedup(FPGA)} = \frac{\#CPU\ runtime(8\ threads)}{\#FPGA\ runtime(N\ PEs)}$$

$$\text{Speedup(CPU)} = \frac{\#CPU\ runtime(8\ threads)}{\#CPU\ runtime(N\ threads)}$$

batch size = 5923

Hardware Resource





- **URAM** is the key to reuse more data on chip (#size of URAM = 8 * #size of BRAM)
- **LUTs and FFs is plenty** compared with RAMs
- Still has potential to **increase the number of PEs**

Outline

- Introduction
- Design Methodology
- Experiment
- **Conclusion**

Conclusion

- Conclusion:
  - ◆ Provide a design methodology to accelerate maze routing on FPGA
  - ◆ Design an efficient data structure, algorithm and hardware implementation
  - ◆ Better scalability than multi-threads software
  - ◆ Up to 3.1x speed-up
- Future Work:
  - ◆ Optimize the performance of routing on the CPU-FPGA system
  - ◆ Process nets with larger size on FPGA

Thanks!
Questions are welcome

Email: jiangx@smail.nju.edu.cn

ICAIS Lab, Nanjing University, China