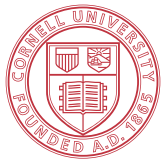
Mapping-aware Logic Synthesis with Parallelized Stochastic Optimization

Zhiru Zhang
School of ECE, Cornell University

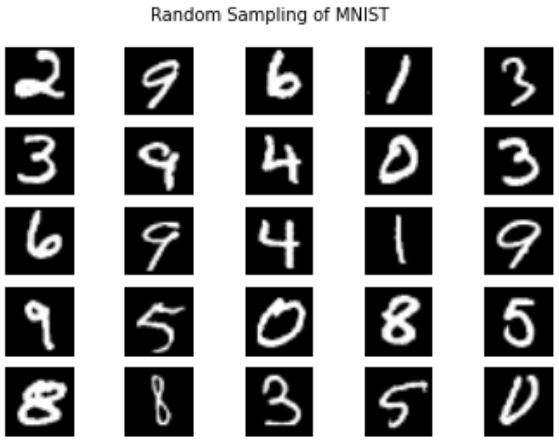
September 29, 2017 @ EPFL



Cornell University

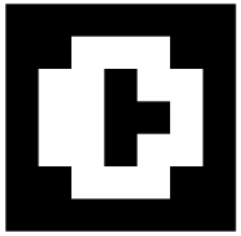
CSL

A Case Study on Digit Recognition




```
0000000
0011100
0110110
0110010
0110110
0011100
0000000
```

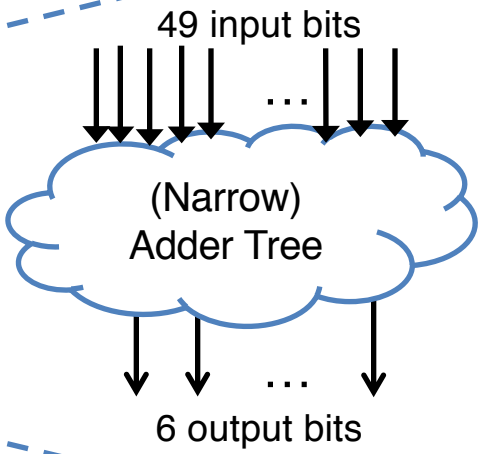
(a) Binary string in 2D array



(b) Binary image

```
bit6 popcount(bit49 digit)
{
  bit6 ones = 0;
  for (i = 0; i < 49; i++)
    ones += digit[i];
  return ones;
}
```

Computes hamming distance between training & test samples

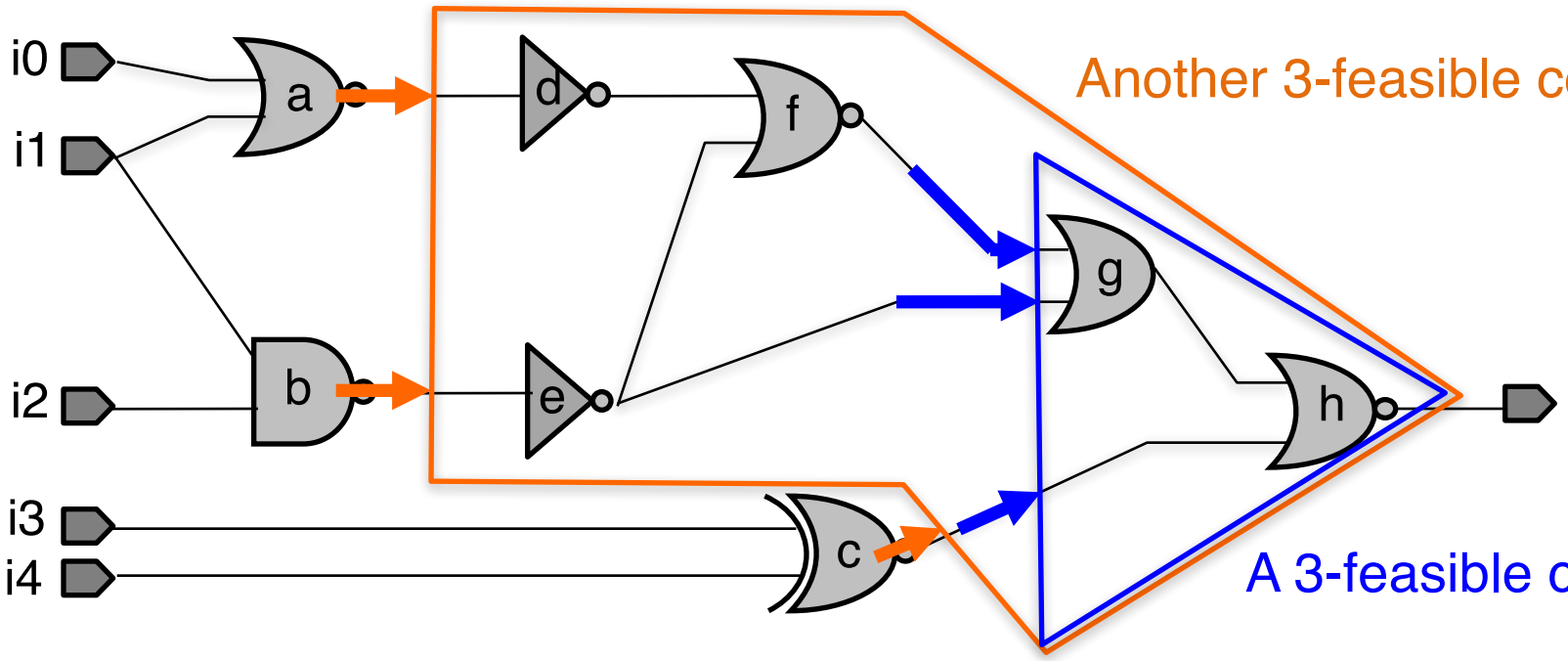


	HLS	Manual
LUTs	2405	1305
FFs	2400	810
BRAMs	60	60

Manual: combinational
HLS tool: 2 cycles

LUT-based Technology Mapping

- Look-up tables (LUTs): the core building blocks of FPGAs
  - A k-LUT can implement any K-input 1-output Boolean function, or any **k-feasible cone** in the logic network



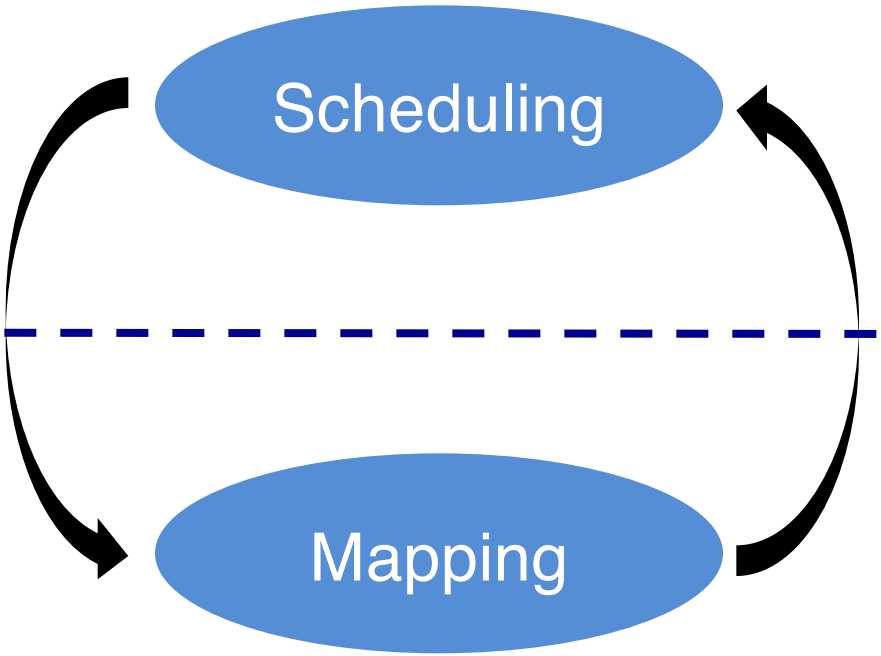
Delay estimation in HLS is inaccurate without considering LUT mapping

Scheduling and Mapping Interdependence

HLS

Determine register boundaries

⇒ Inaccurate delay model due to lack of mapping awareness

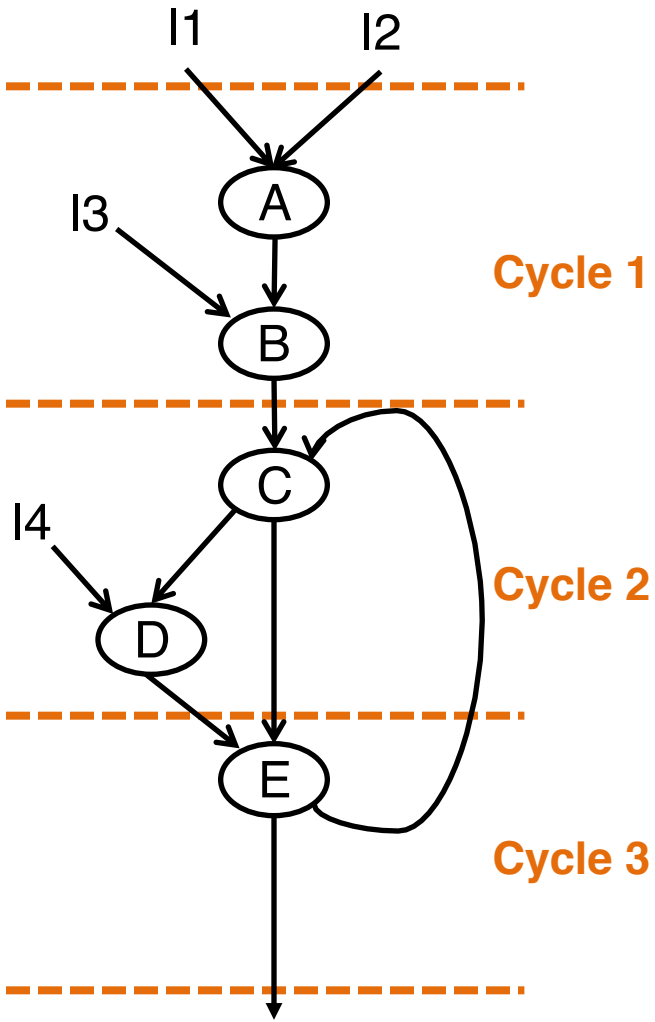


Post-RTL Flow

Determine LUT network

⇒ More realistic delay, but cannot change register boundaries

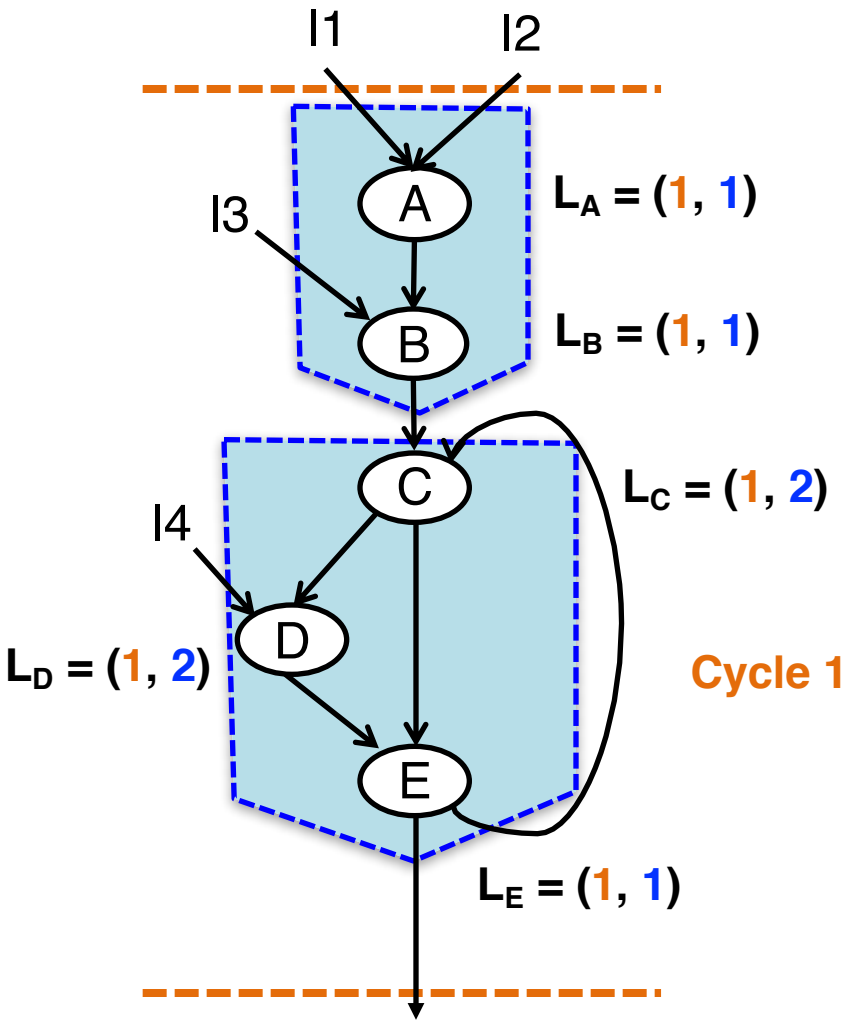
Mapping-Aware Scheduling [FPGA'15]



Conventional schedule

(5ns cycle time target)

Latency = 3 cycles



Mapping-aware schedule

(4-input LUT)

Latency = 1 cycle

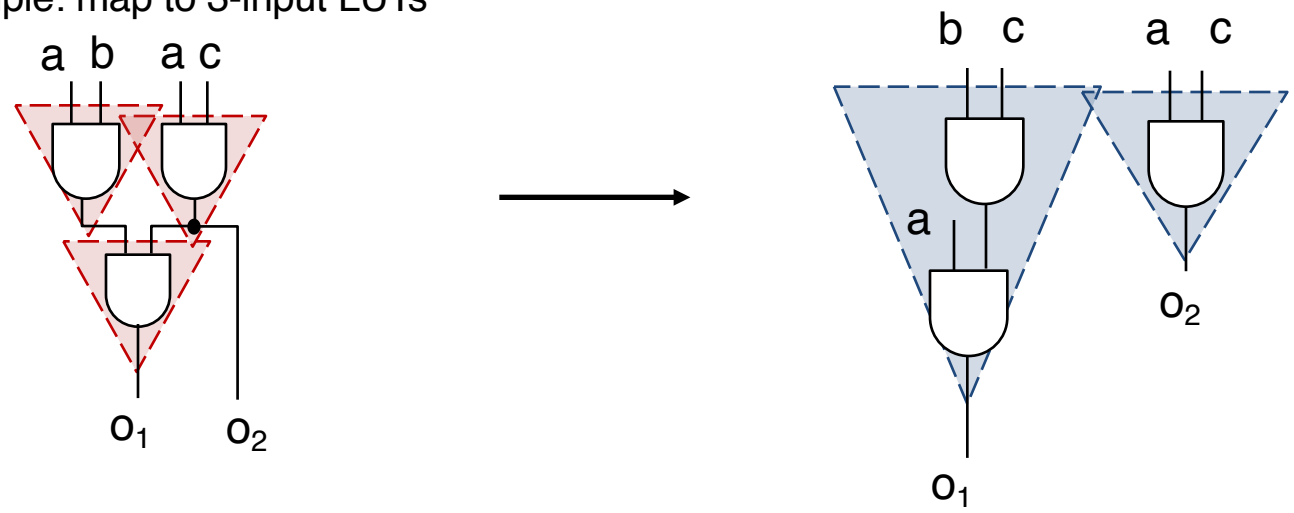
M. Tan, S. Dai, U. Gupta, Z. Zhang, **Mapping-Aware Constrained Scheduling for LUT-Based FPGAs**, FPGA'2015.

What about Post-RTL Synthesis?

- Case 1: Min-area mapping without logic restructuring
  - Already NP-hard [1]

- Case 2: with logic restructuring ⟵ Focus of this talk
  - Even harder to find optimal solution
  - Existing approach: heuristically transform logic network for better mapping quality

Example: map to 3-input LUTs



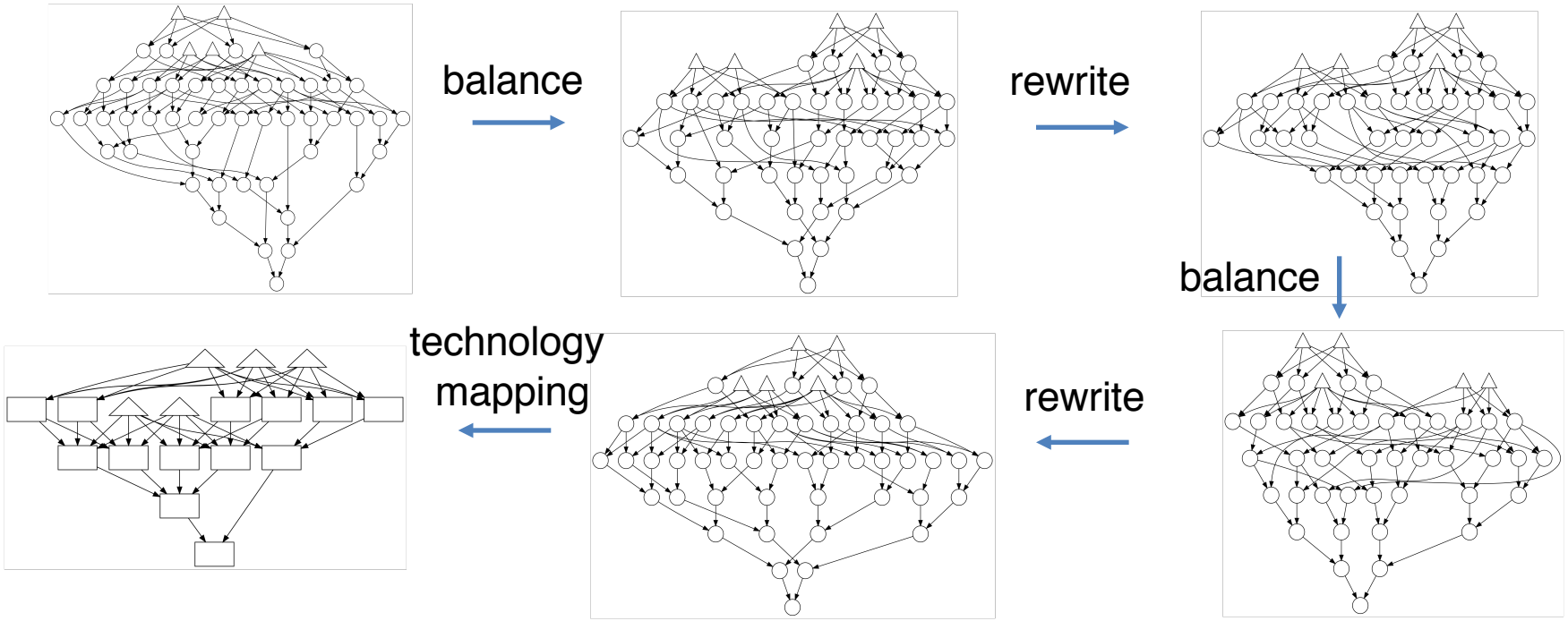
[1] Farrahi and Sarrafzadeh, TCAD'02

Typical Pre-mapping Transformation Sequence

- A typical area-minimizing script in ABC:

balance → rewrite → balance → rewrite

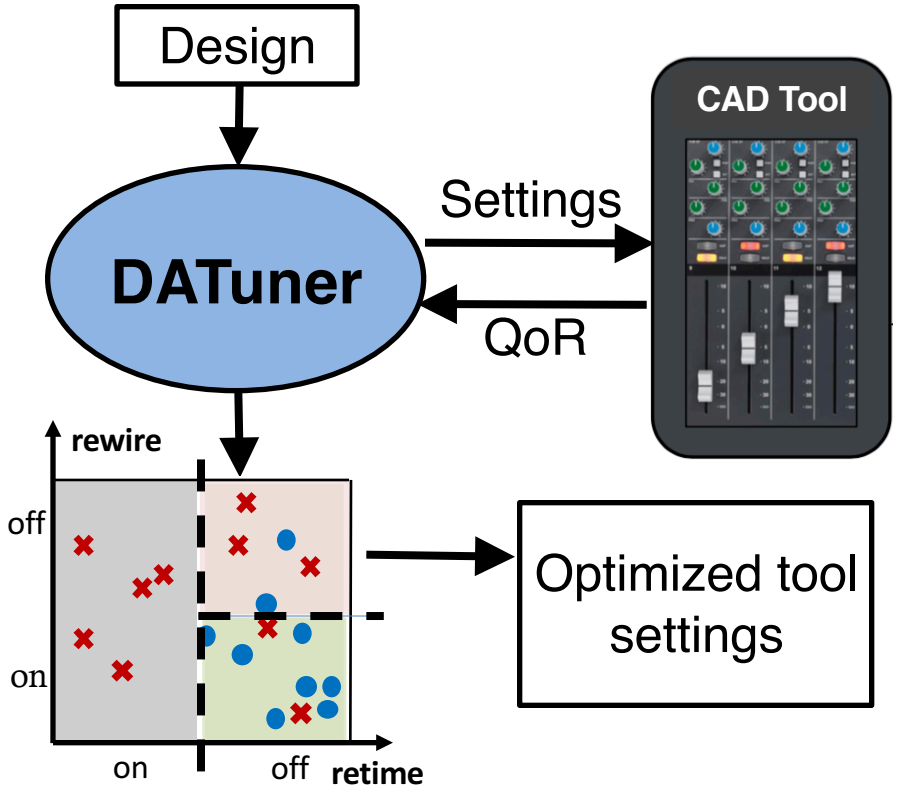
Initial and-inverter graph for *xor5*



A predetermined technology-independent optimization sequence

Autotuning Logic Synthesis Tool

- Applying DATuner, a distributed autotuning framework, to auto determine the logic transformation sequence

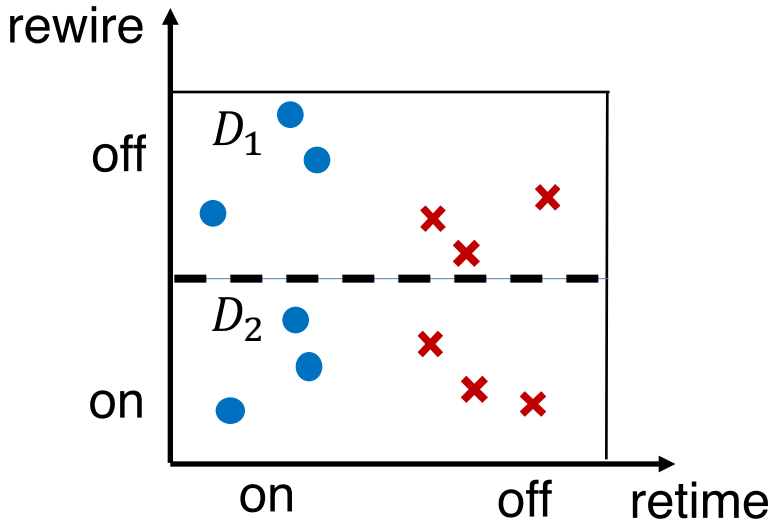


github.com/cornell-zhang/datuner

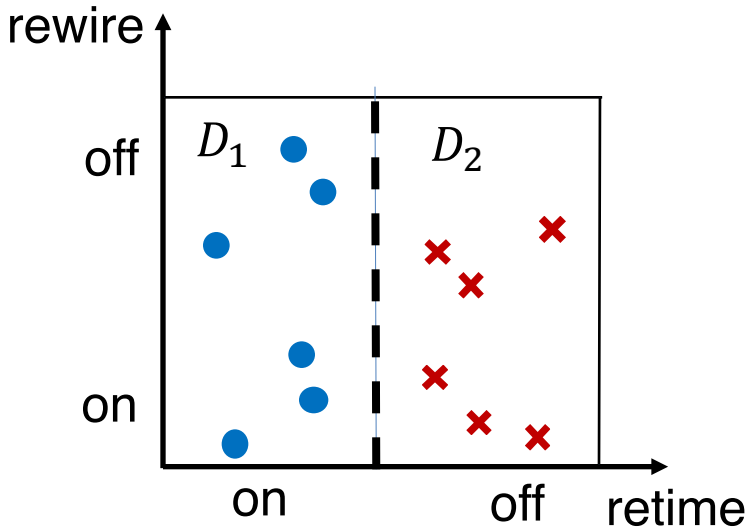
C. Xu, G. Liu, R. Zhao, S. Yang, G. Luo, and Z. Zhang, **A Parallel Bandit-Based Approach for Autotuning FPGA Compilation**, FPGA'2017.

DATuner: Dynamic Solution Space Partitioning

- Separating promising from unpromising subspaces
  - Guided by information gain derived from QoR of known samples



Partition A (bad)



Partition B (good)

$$H(D_0) = -\frac{6}{12}\times\log\left(\frac{6}{12}\right)-\frac{6}{12}\times\log\left(\frac{6}{12}\right) = 0.3 \qquad (D_0 = D_1 \cup D_2)$$

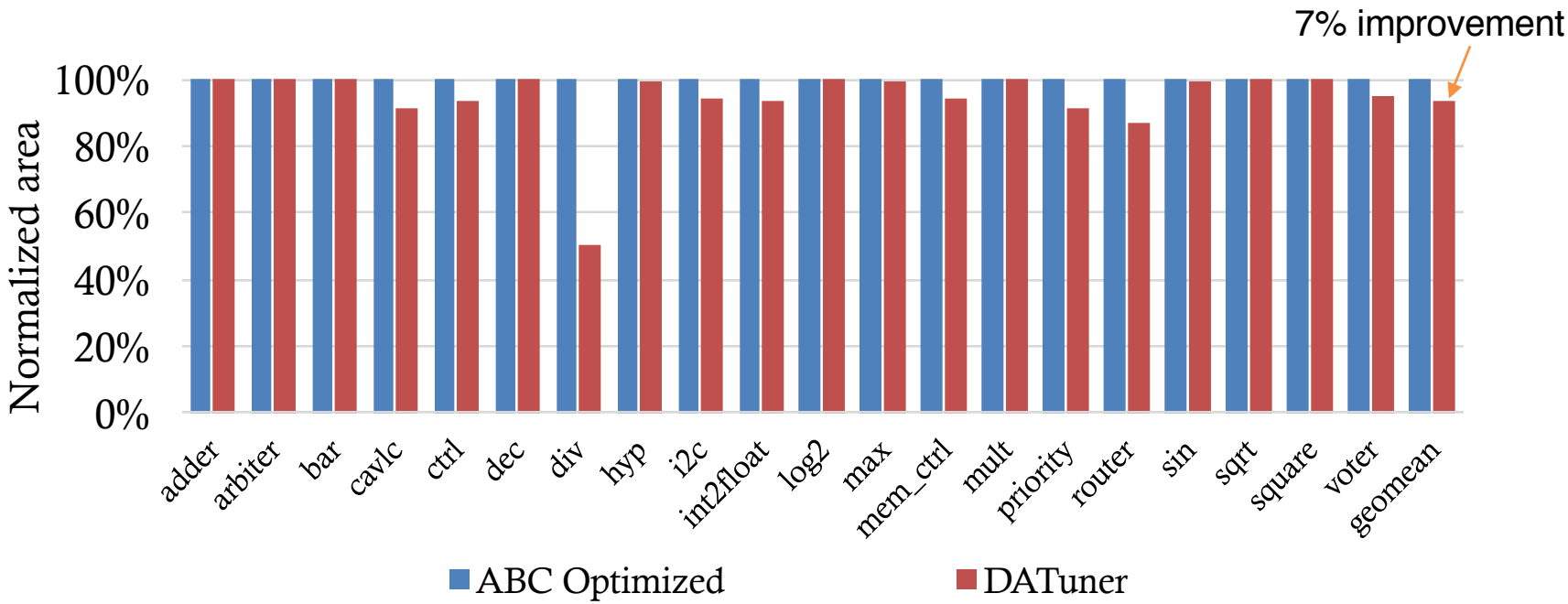
$H(D_1) = H(D_2) = 0.3$

Information gain = $0.3 - \frac{1}{2}(0.3 + 0.3) = 0$

$H(D_1) = H(D_2) = 0$

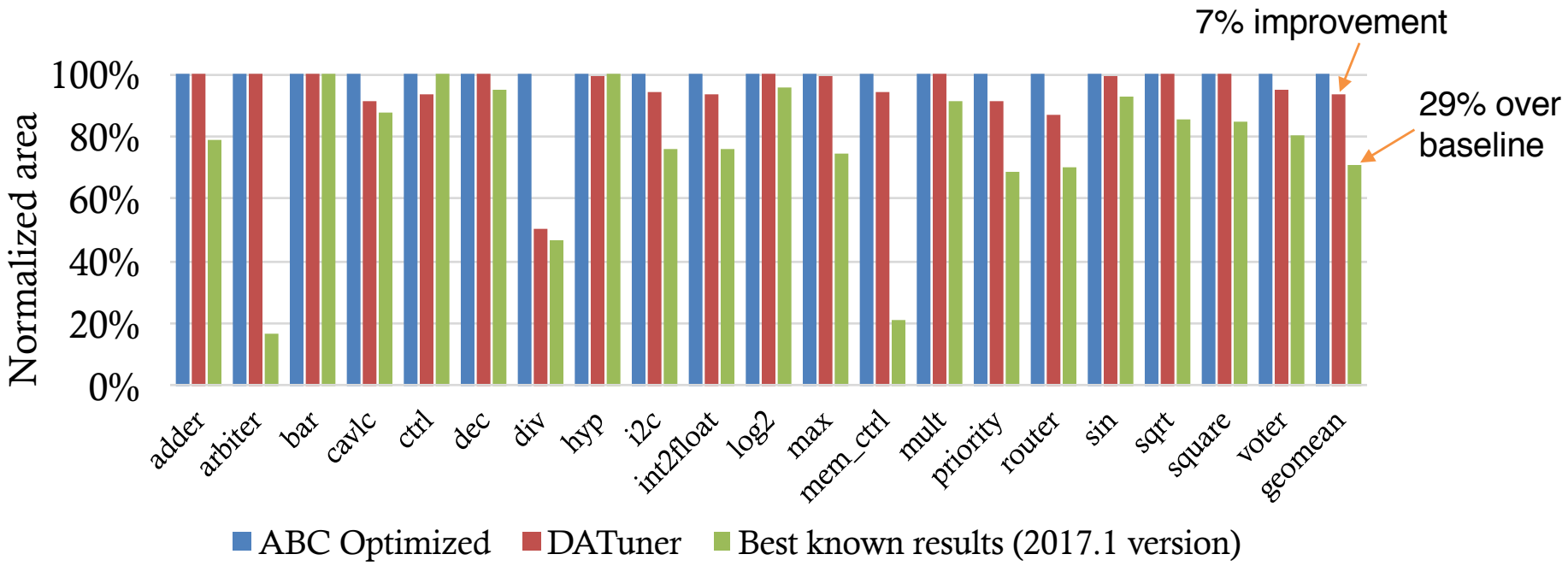
Information gain = $0.3 - \frac{1}{2}(0 + 0) = 0.3$

Autotuning vs. ABC: Unconstrained Area Minimization



- ABC Optimized: designs optimized with *compress2rs* script
- DATuner: a budget of 128 ABC runs across 4 machines
- EPFL benchmarks: http://lsi.epfl.ch/benchmarks

Autotuning vs. Best Known Records (v2017.1)

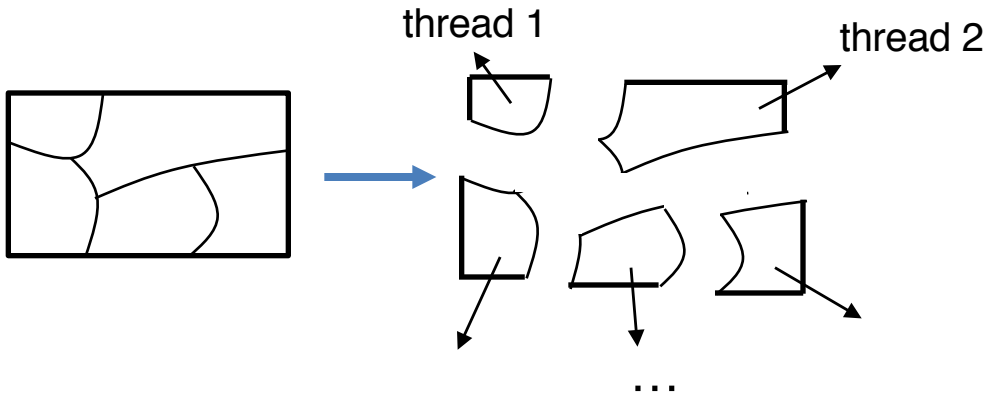


- ABC Optimized: designs optimized with *compress2rs* script
- DATuner: a budget of 128 ABC runs across 4 machines
- EPFL benchmarks: http://lsi.epfl.ch/benchmarks
  - Best known results: from EPFL record, version 2017.1

PIMap: Parallelized Mapping-Aware Logic Synthesis [FPGA'17]

- Mapping-guided logic transformations
  - Iteratively improve area

- Effective partitioning and parallelization technique
  - Improve both runtime and design quality



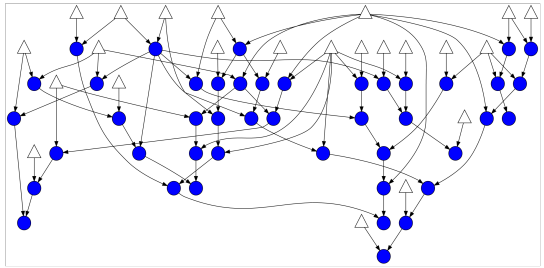
G. Liu and Z. Zhang, **A Parallelized Iterative Improvement Approach to Area Optimization for LUT-Based Technology Mapping**, FPGA'2017.

PIMap Technique: Iterative Area Minimization

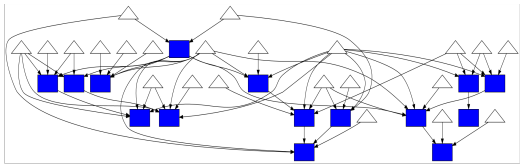
Use mapping result to guide randomly proposed logic transformations

PIMap Technique: Iterative Area Minimization

Use mapping result to guide randomly proposed logic transformations

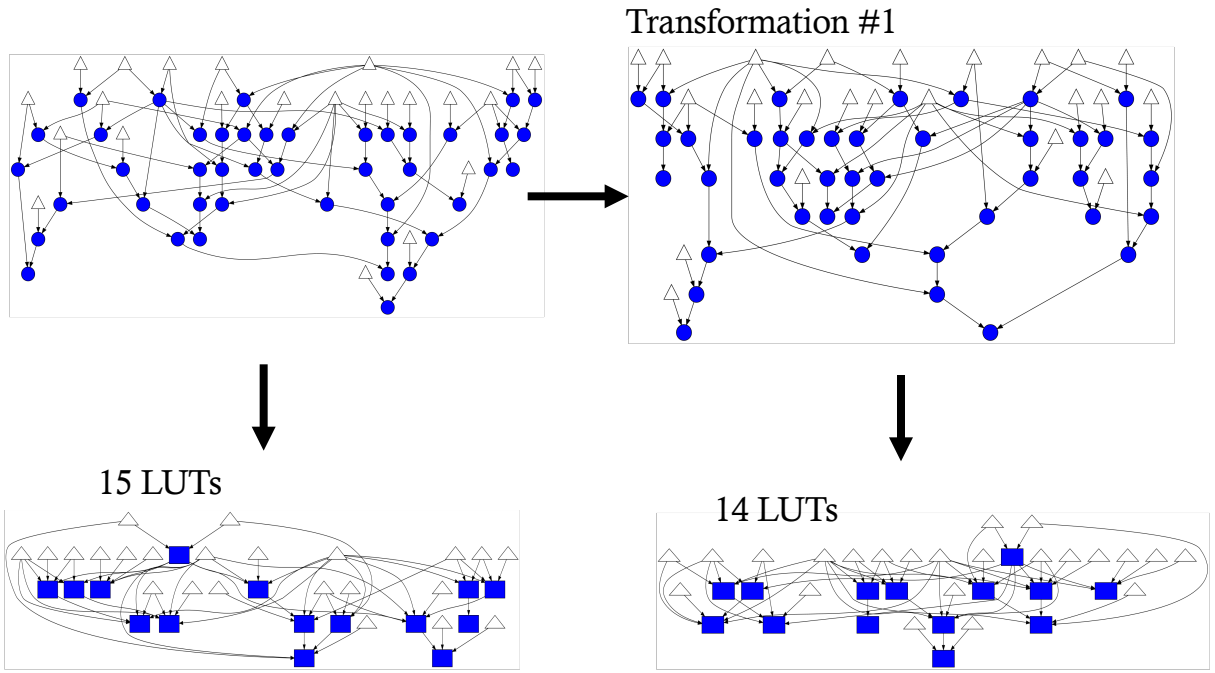


15 LUTs



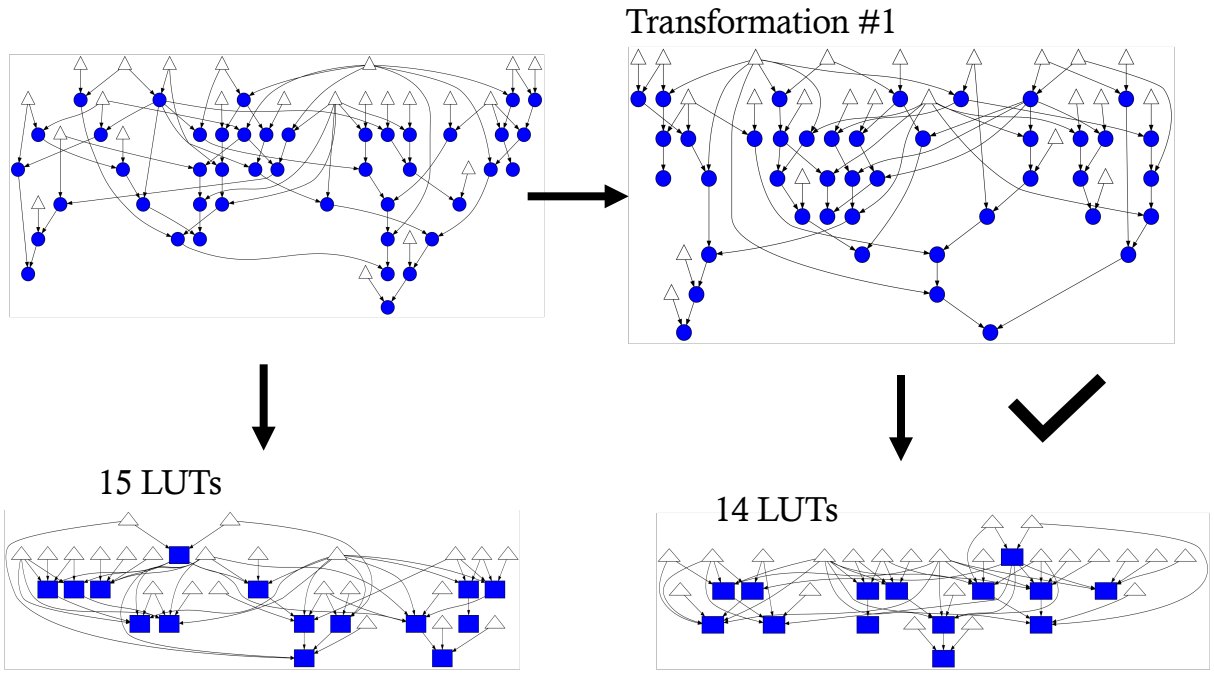
PIMap Technique: Iterative Area Minimization

Use mapping result to guide randomly proposed logic transformations



PIMap Technique: Iterative Area Minimization

Use mapping result to guide randomly proposed logic transformations



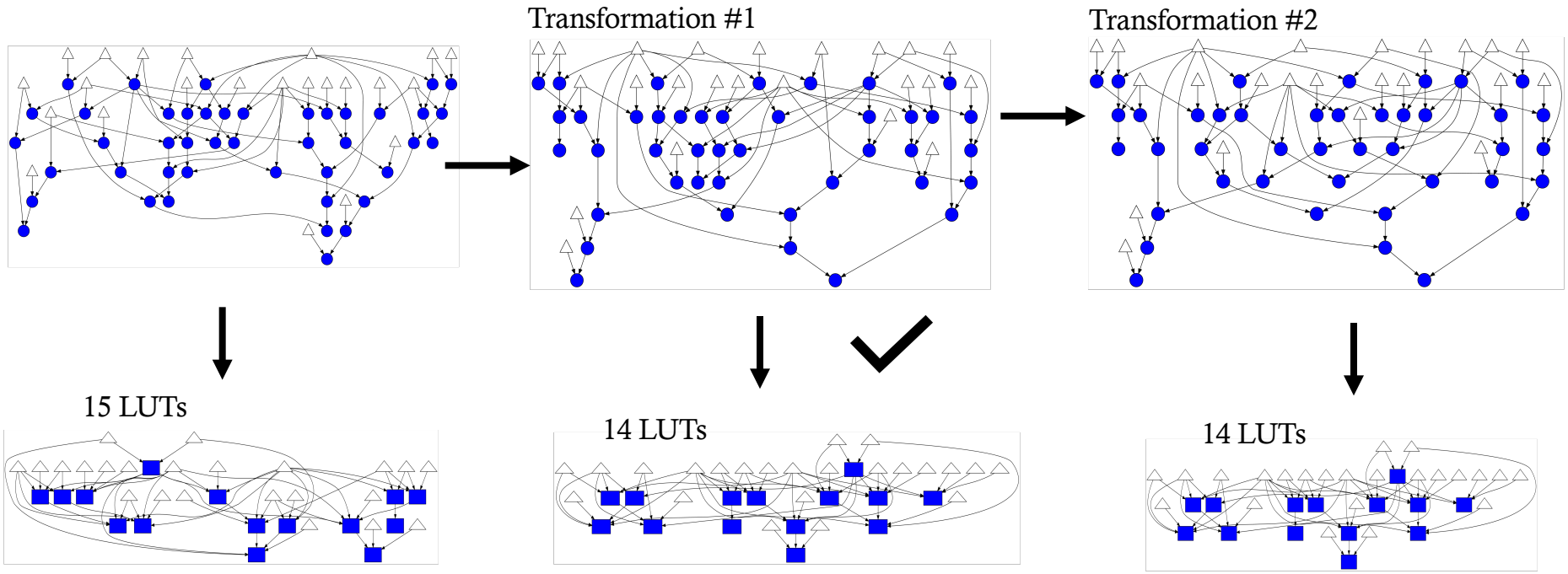
Metropolis-Hastings algorithm[1]:

Accept current transformation if $rand(0,1) < \exp(-\gamma \frac{N_{LUT_{\downarrow}new}}{N_{LUT_{\downarrow}old}})$

[1] Hastings, Biometrika'70

PIMap Technique: Iterative Area Minimization

Use mapping result to guide randomly proposed logic transformations



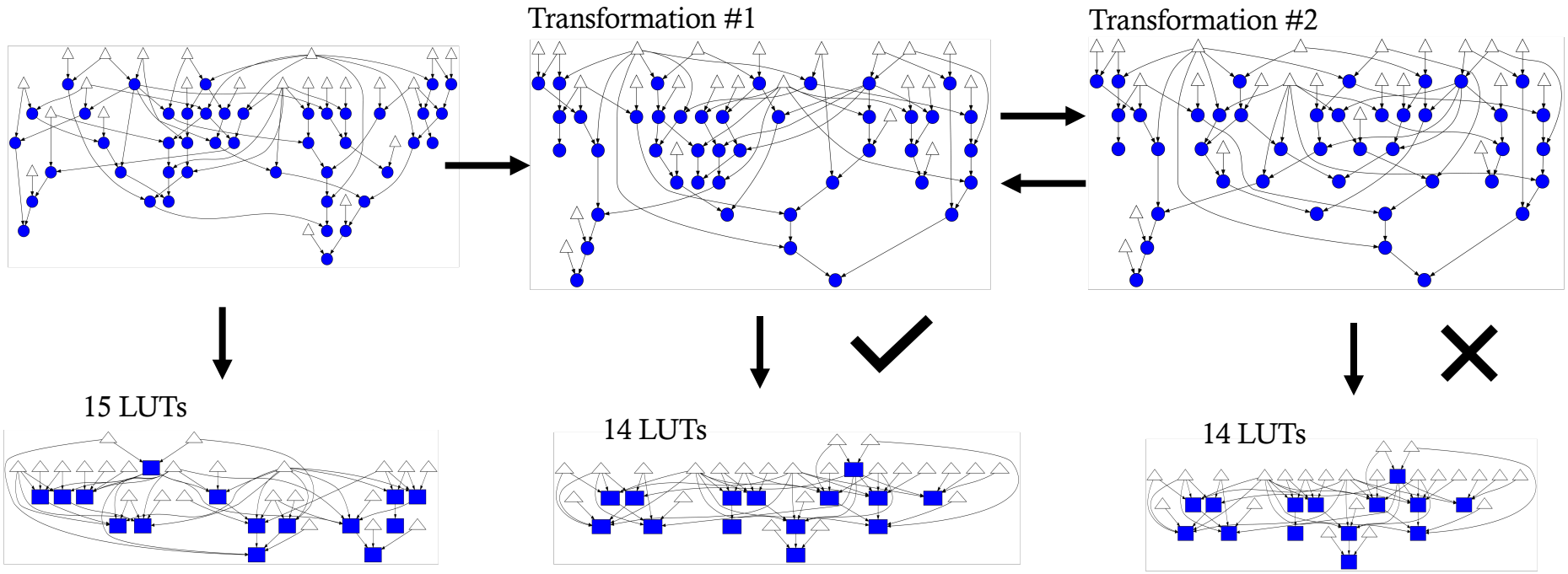
Metropolis-Hastings algorithm[1]:

Accept current transformation if $rand(0,1) < \exp(-\gamma \frac{N_{LUT_{\downarrow}new}}{N_{LUT_{\downarrow}old}})$

[1] Hastings, Biometrika'70

PIMap Technique: Iterative Area Minimization

Use mapping result to guide randomly proposed logic transformations



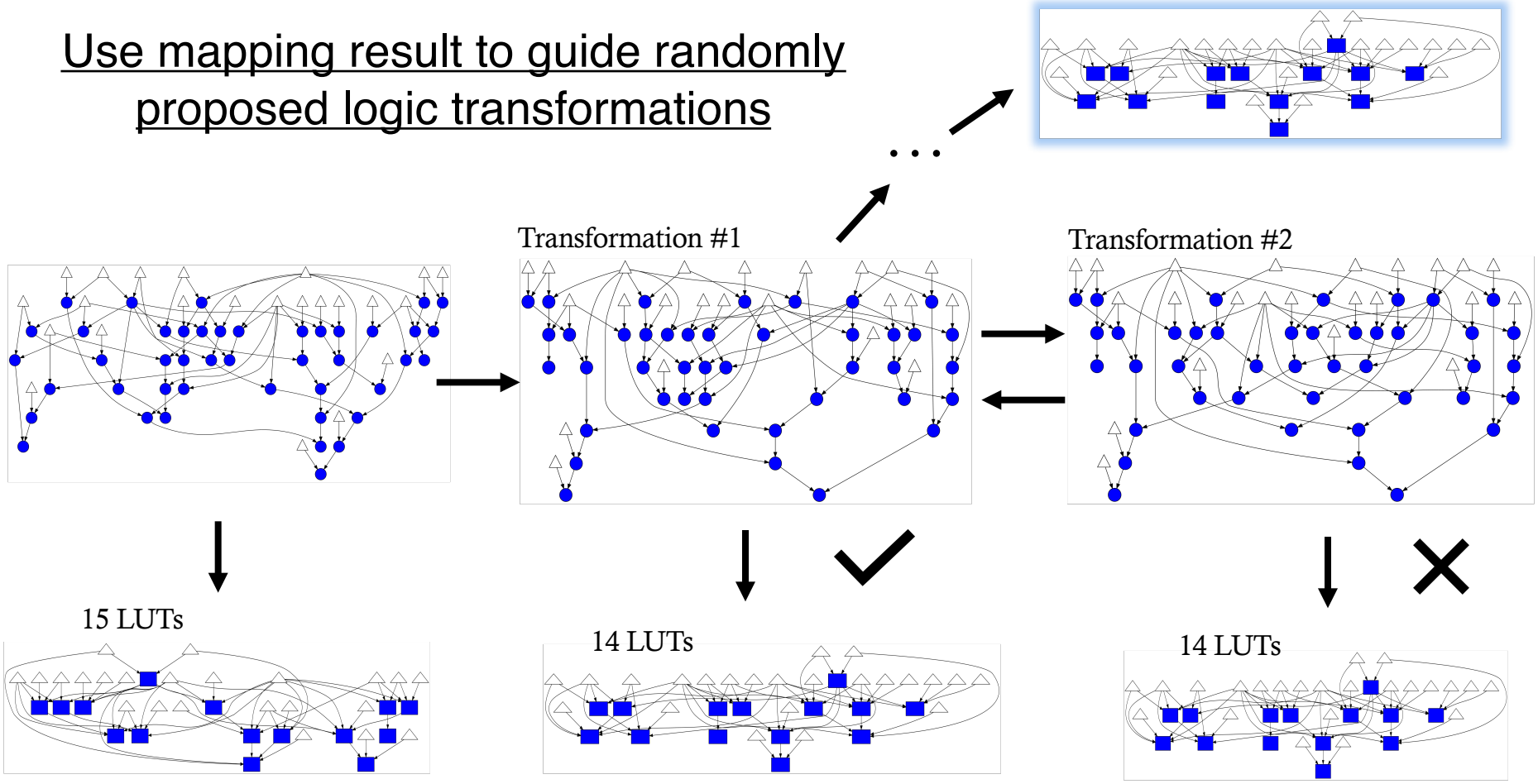
Metropolis-Hastings algorithm[1]:

Accept current transformation if $rand(0,1) < \exp(-\gamma \frac{N_{LUT_{\downarrow}new}}{N_{LUT_{\downarrow}old}})$

[1] Hastings, Biometrika'70

PIMap Technique: Iterative Area Minimization

Use mapping result to guide randomly proposed logic transformations



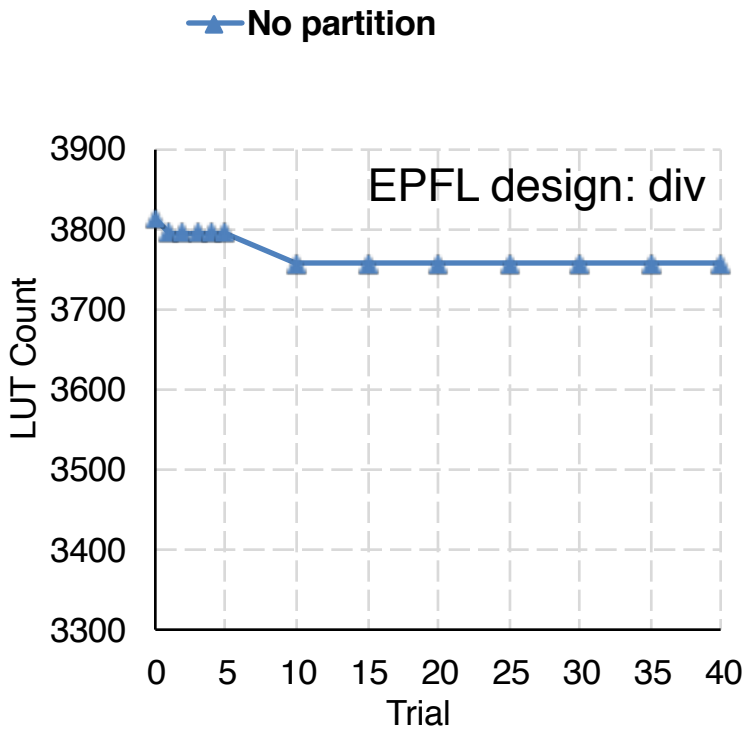
Metropolis-Hastings algorithm[1]:

Accept current transformation if $rand(0,1) < \exp(-\gamma \frac{N_{LUT_{\downarrow}new}}{N_{LUT_{\downarrow}old}})$

[1] Hastings, Biometrika'70

Need for Partitioning

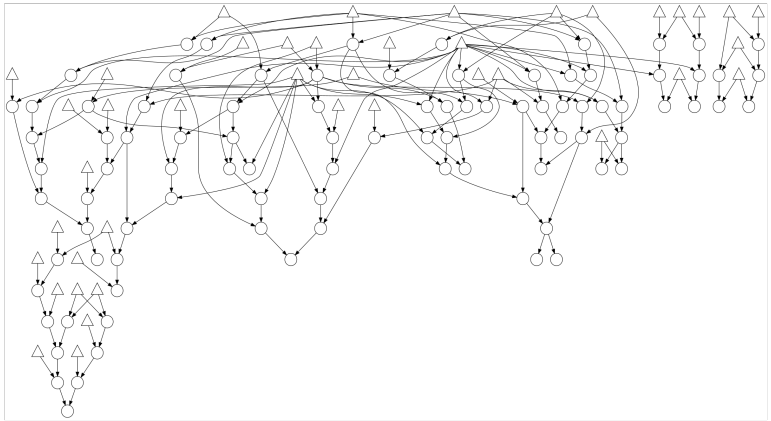
- Without partitioning
  - Long runtime per trial
  - Easily stuck at local minimum



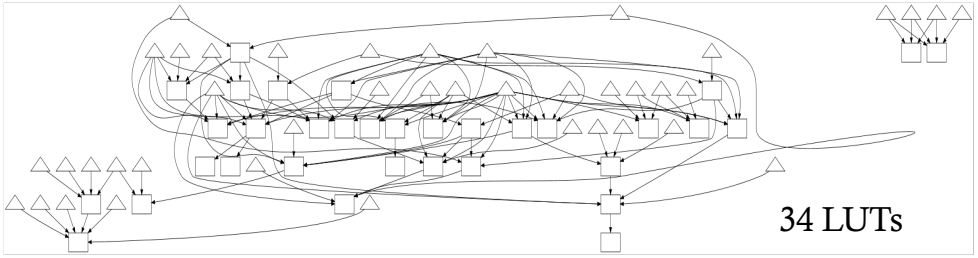
PIMap Technique: Partitioning and Parallelization

Initial mapping to LUT Subgraph extraction Iterative area minimization Recombine subgraphs

AIG of design b9

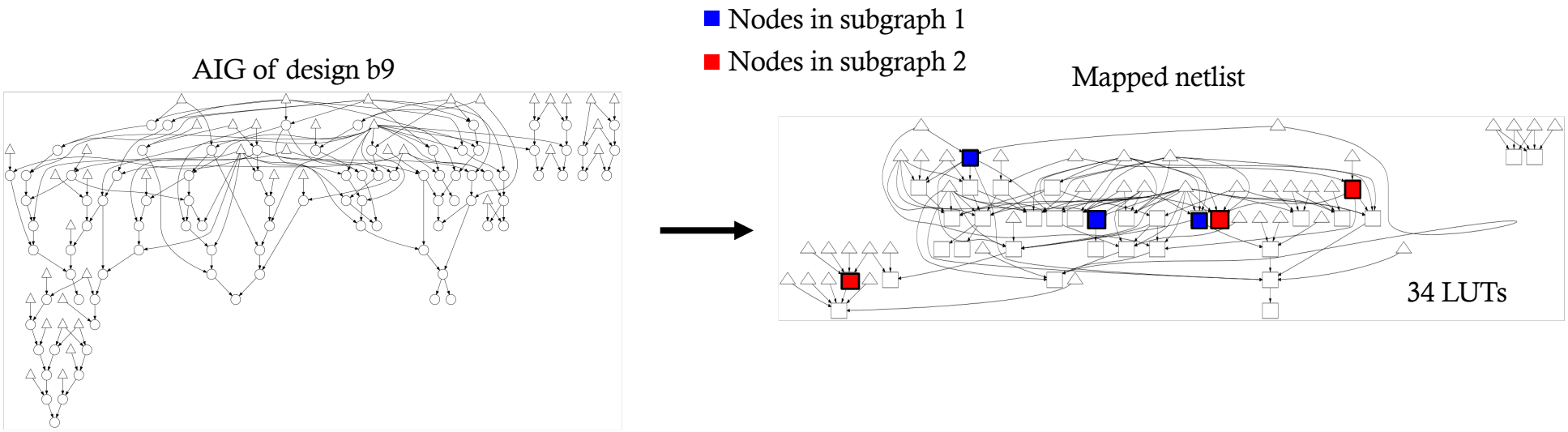


Mapped netlist



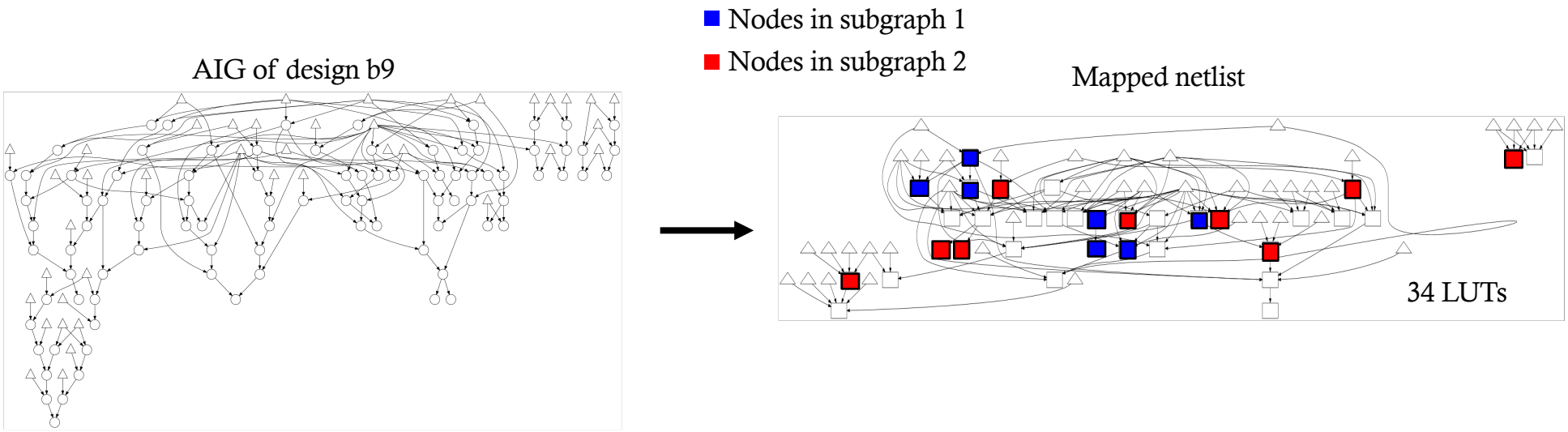
PIMap Technique: Partitioning and Parallelization

Initial mapping to LUT **Subgraph extraction** Iterative area minimization Recombine subgraphs



PIMap Technique: Partitioning and Parallelization

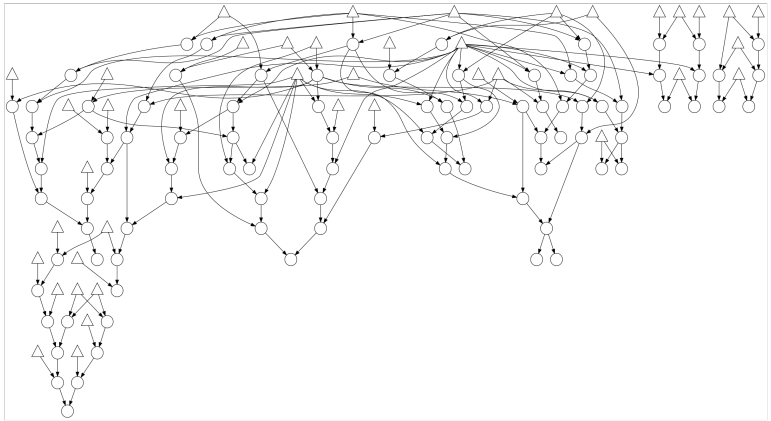
Initial mapping to LUT **Subgraph extraction** Iterative area minimization Recombine subgraphs



PIMap Technique: Partitioning and Parallelization

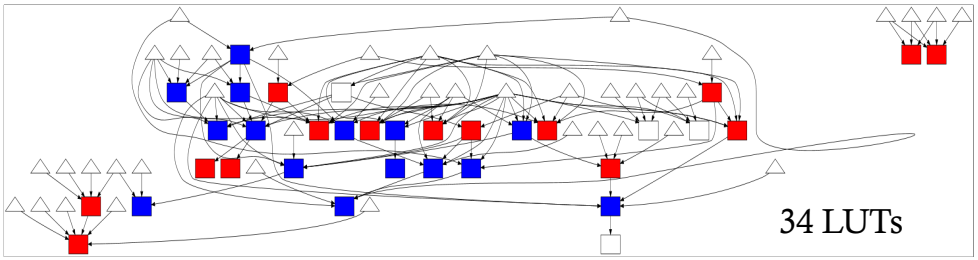
Initial mapping to LUT **Subgraph extraction** Iterative area minimization Recombine subgraphs

AIG of design b9



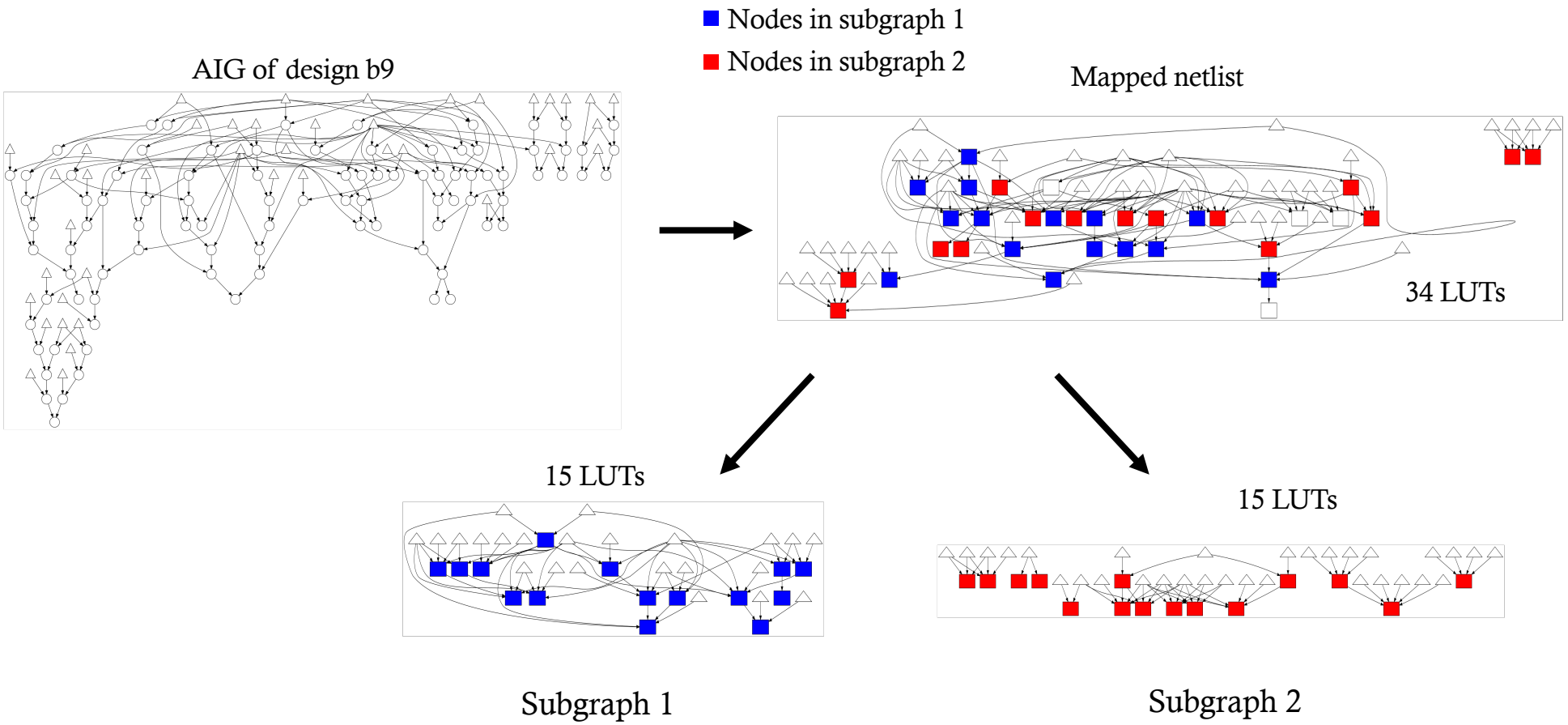
Nodes in subgraph 1
Nodes in subgraph 2

Mapped netlist



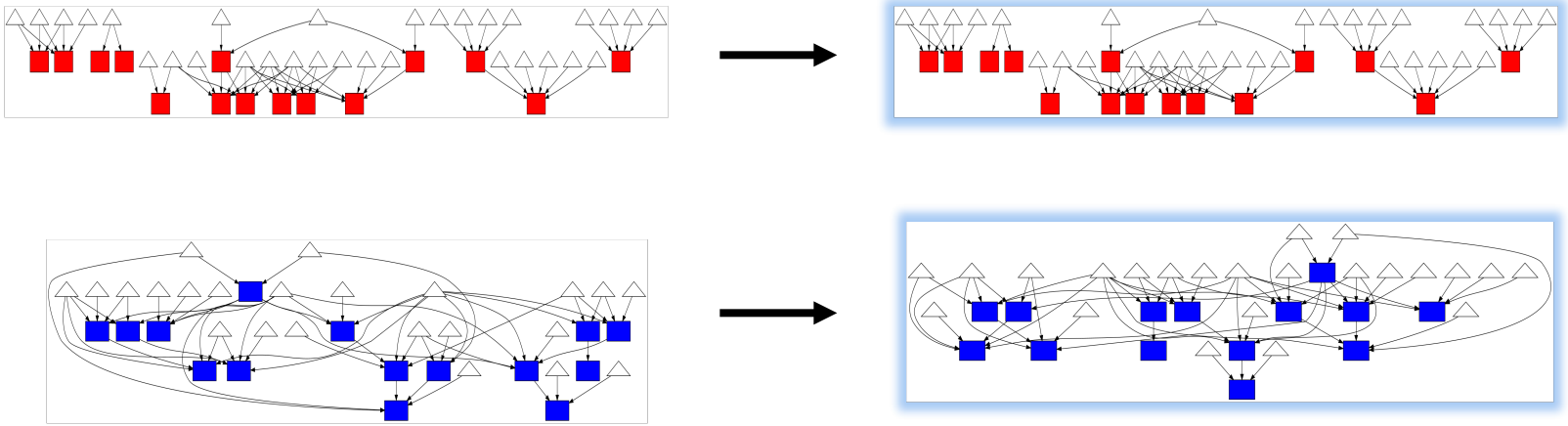
PIMap Technique: Partitioning and Parallelization

Initial mapping to LUT | **Subgraph extraction** | Iterative area minimization | Recombine subgraphs



PIMap Technique: Partitioning and Parallelization

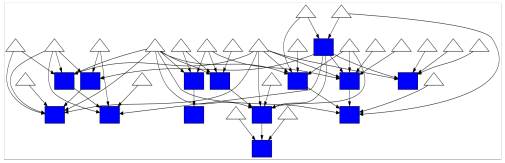
Initial mapping to LUT Subgraph extraction **Iterative area minimization** Recombine subgraphs



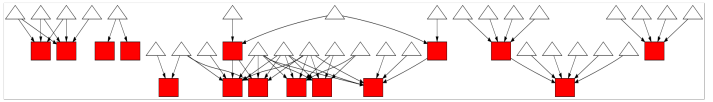
PIMap Technique: Partitioning and Parallelization

Initial mapping to LUT Subgraph extraction Iterative area minimization **Recombine subgraphs**

14 LUTs

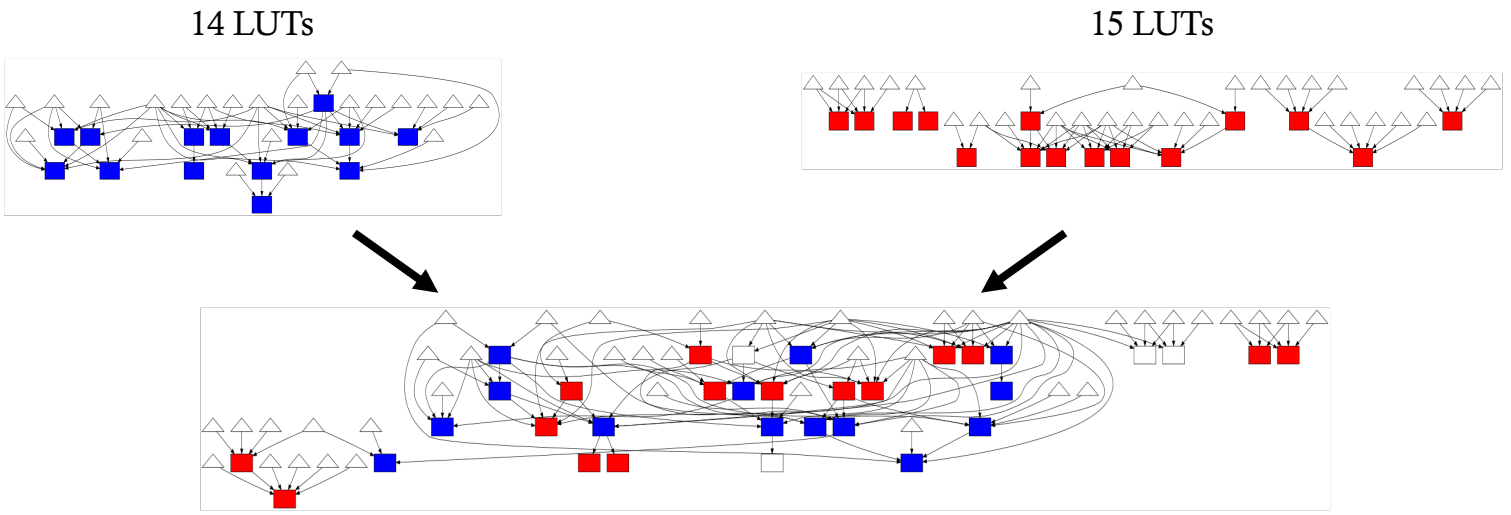


15 LUTs



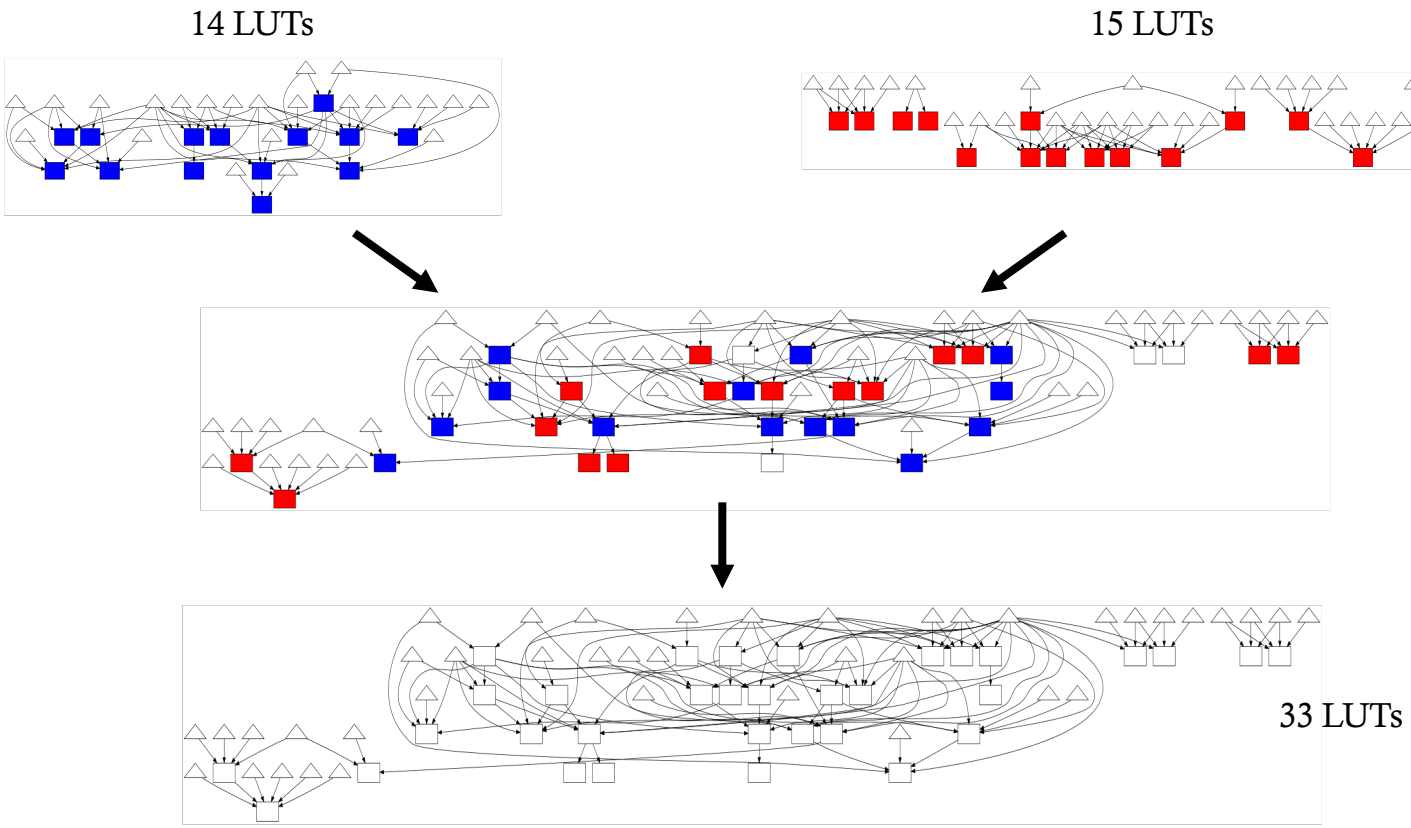
PIMap Technique: Partitioning and Parallelization

Initial mapping to LUT Subgraph extraction Iterative area minimization **Recombine subgraphs**



PIMap Technique: Partitioning and Parallelization

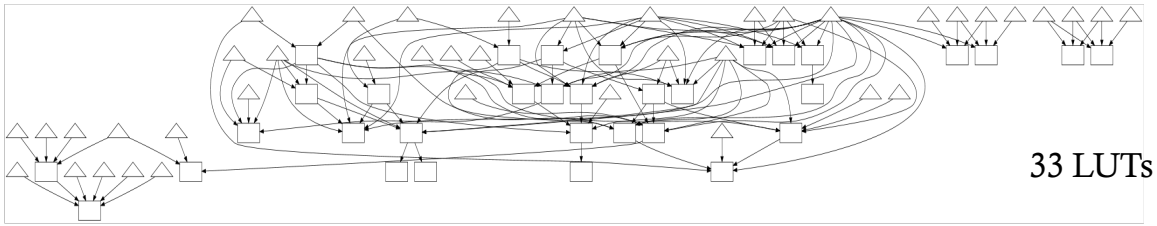
Initial mapping to LUT Subgraph extraction Iterative area minimization **Recombine subgraphs**



PIMap Technique: Repartition

Initial mapping to LUT Subgraph extraction Iterative area minimization Recombine subgraphs

Optimized design after trial 1

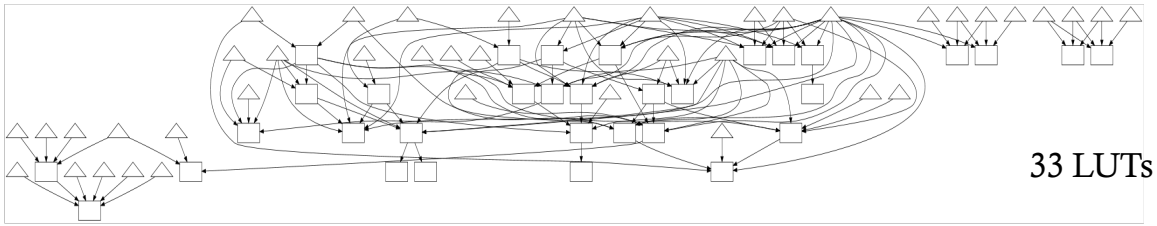


Repartition using different seeds

PIMap Technique: Repartition

Initial mapping to LUT Subgraph extraction Iterative area minimization Recombine subgraphs

Optimized design after trial 1

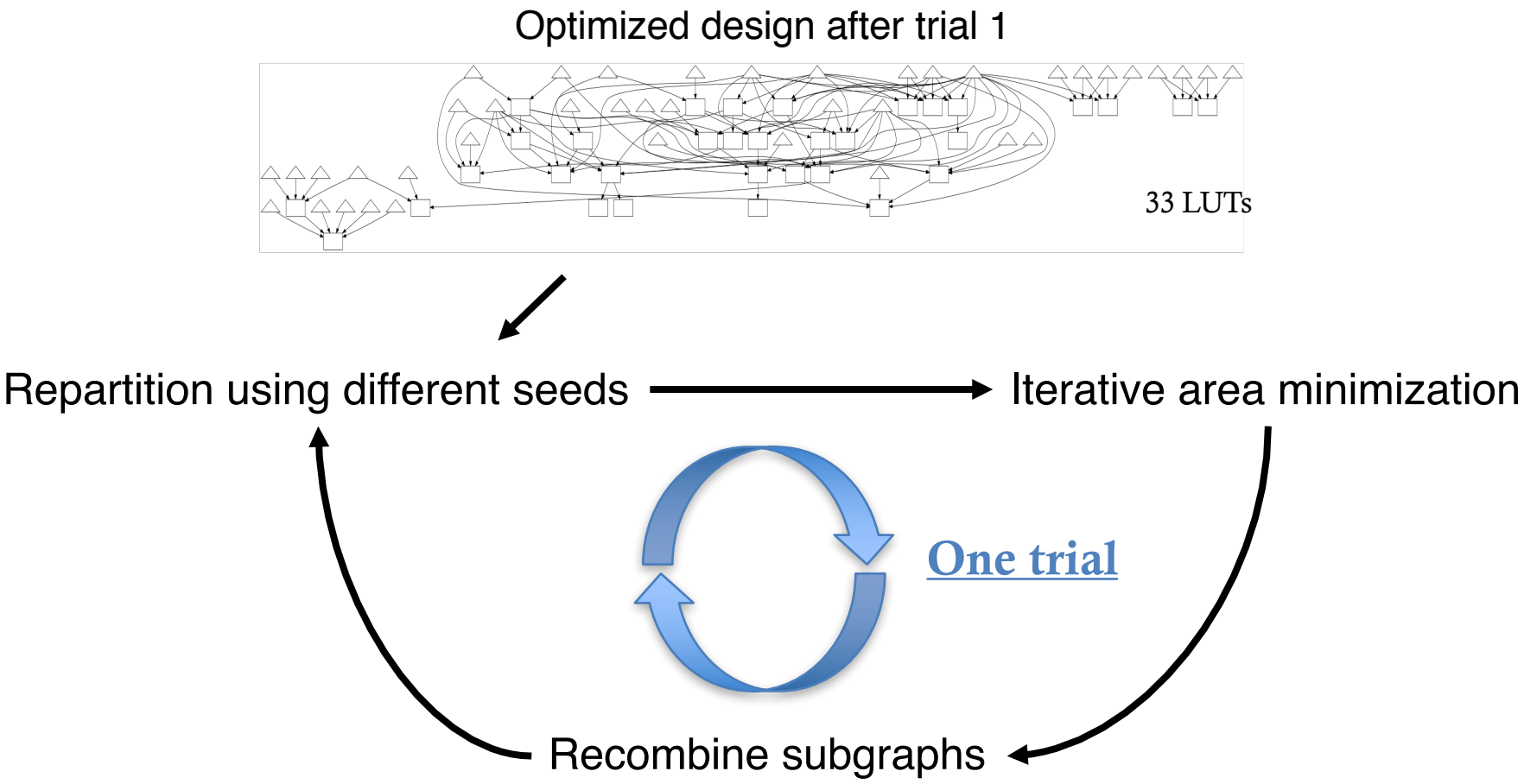


Repartition using different seeds → Iterative area minimization

Recombine subgraphs

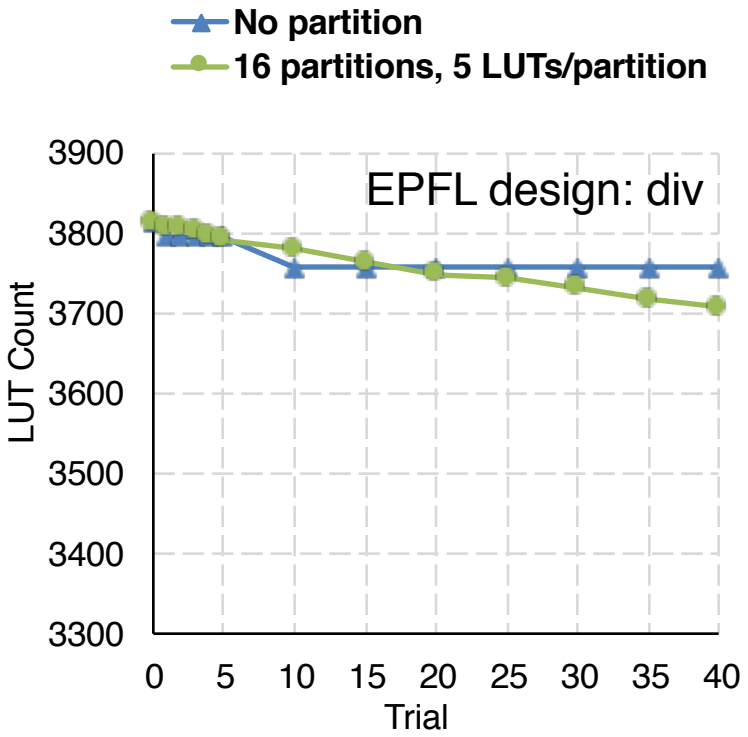
PIMap Technique: Repartition

Initial mapping to LUT Subgraph extraction Iterative area minimization Recombine subgraphs



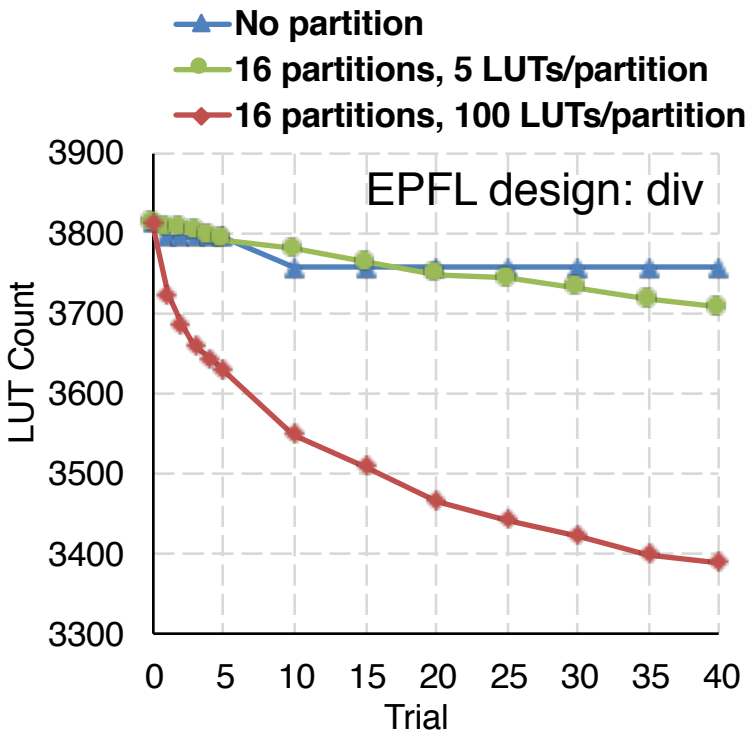
Partitioning Schemes

- Without partitioning
  - Long runtime per trial
  - Easily stuck in local optima
- Fine-grained partition
  - Bear a similarity to exact synthesis
  - Fast runtime per trial
  - But slow progress overall



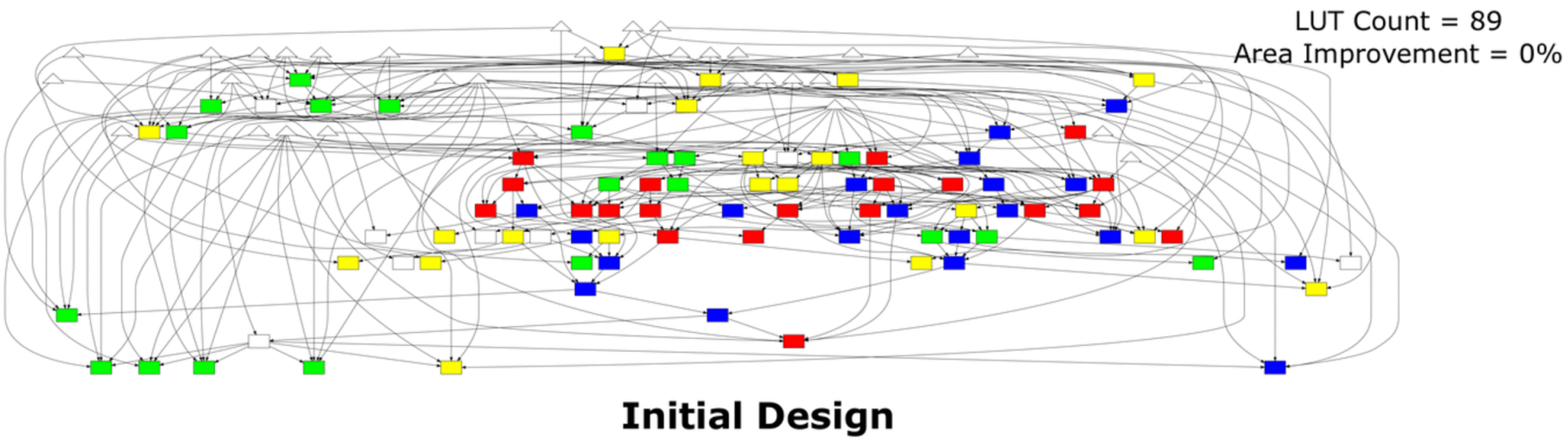
Partitioning Schemes

- Without partitioning
  - Long runtime per trial
  - Easily stuck in local optima
- Fine-grained partition
  - Bear a similarity to exact synthesis
  - Fast runtime per trial
  - But slow progress overall
- **Coarse-grained partition**
  - Balance runtime and solution quality
  - Repartition between trials to further improve quality



PIMap Overall Flow

Design C1908 from the MCNC benchmark suite
5 trials in total

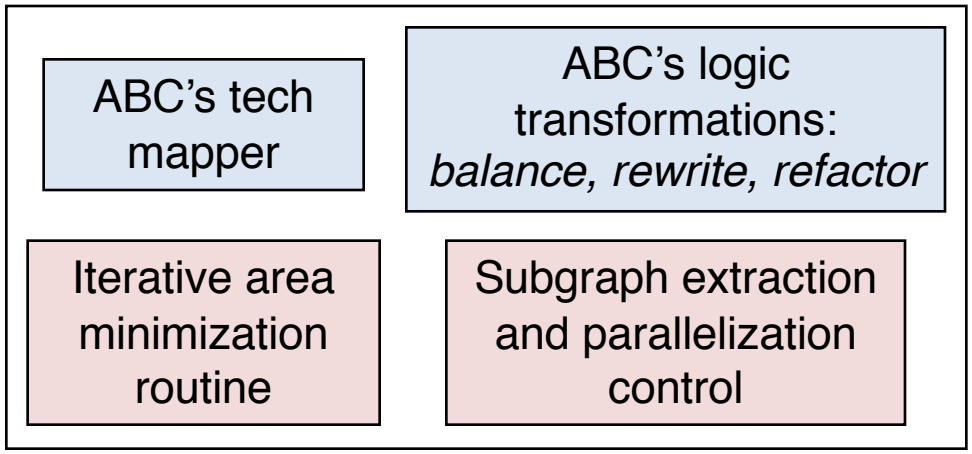


Observations:

1. Partition boundaries vary between trials
   → Uncover better structure
2. Overall network structure differ significantly between trials
   → Discover a wide range of designs

Experimental Setup

PIMap toolchain



Benchmarks

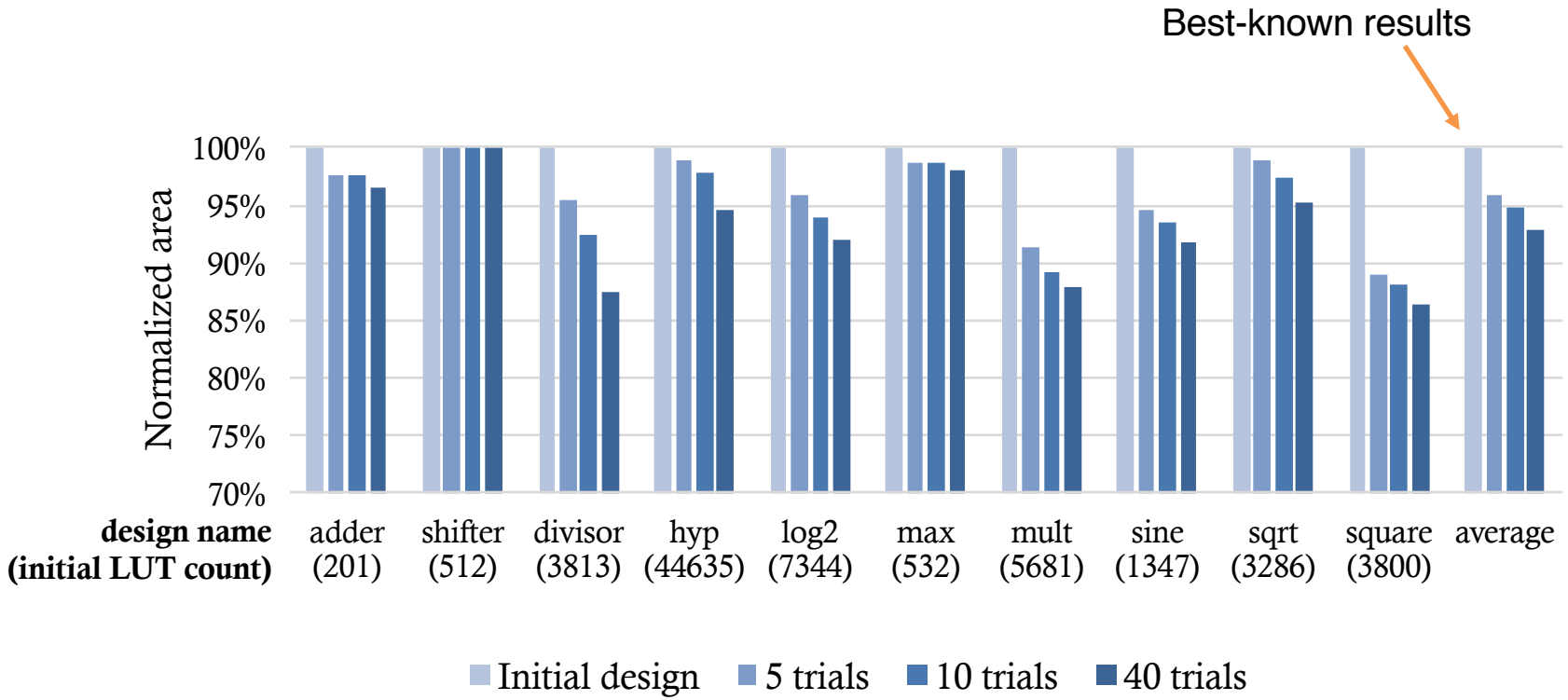
Benchmark	Initial design
10 largest MCNC designs [1]	pre-synthesized using ABC's *compress2rs* script
EPFL arithmetic designs [2]	best-known mapping designs [2]

[1] Yang, MCNC'91
[2] Amarù, et al., http://lsi.epfl.ch/benchmarks

Setup

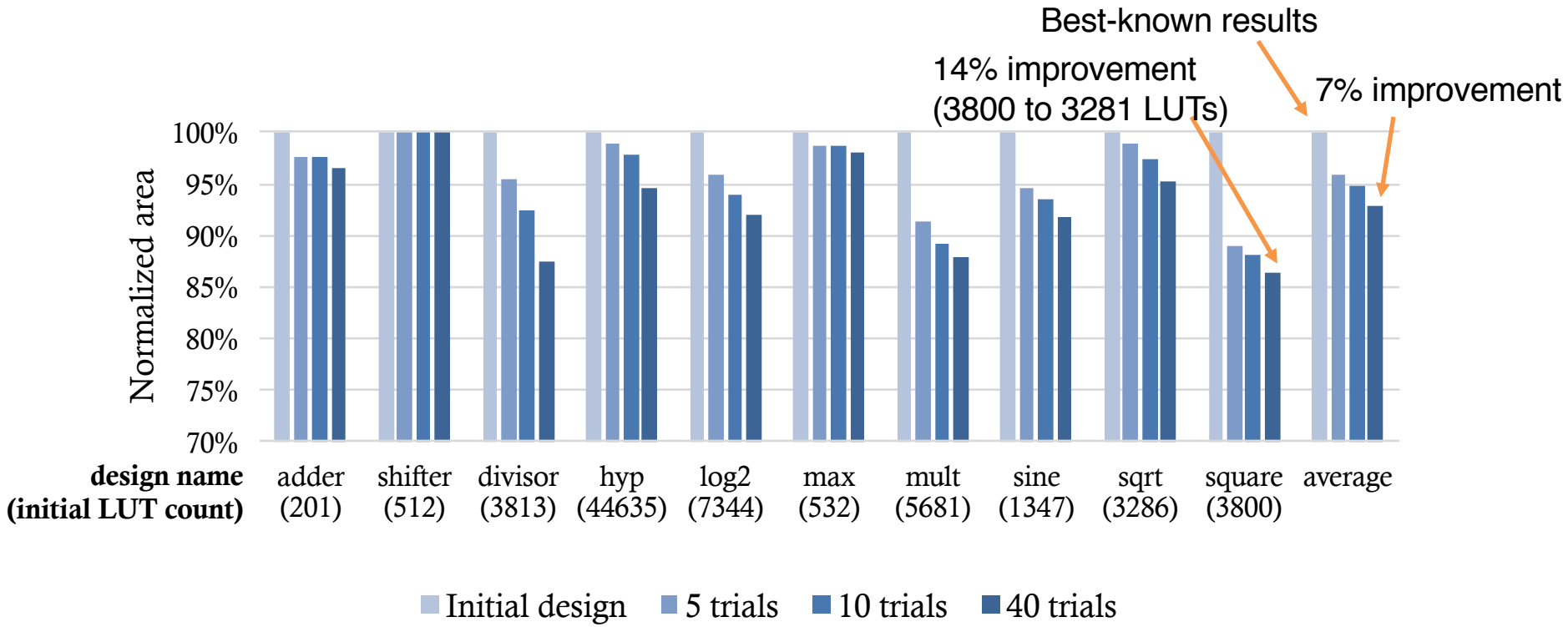
Configuration	40 trials, 100 iterations of area minimization per trial
Partitioning	up to 16 subgraphs, each with up to 100 LUTs
Computing resource	up to 8 machines, each with a quad-core Xeon processor

Area Minimization Results



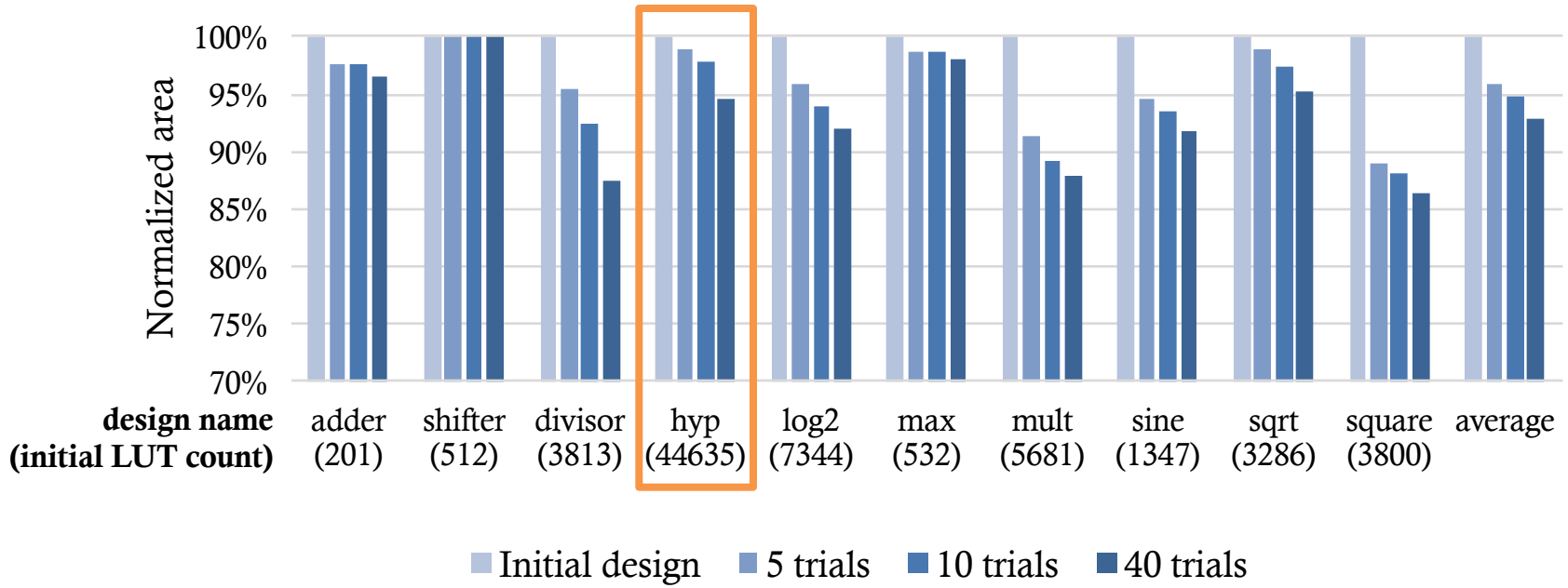
- Initial design: best-known results from EPFL record

Area Minimization Results



- Initial design: best-known results from EPFL record
- Area improvements
  - EPFL: 7% on average, up to 14%

Area Minimization Results

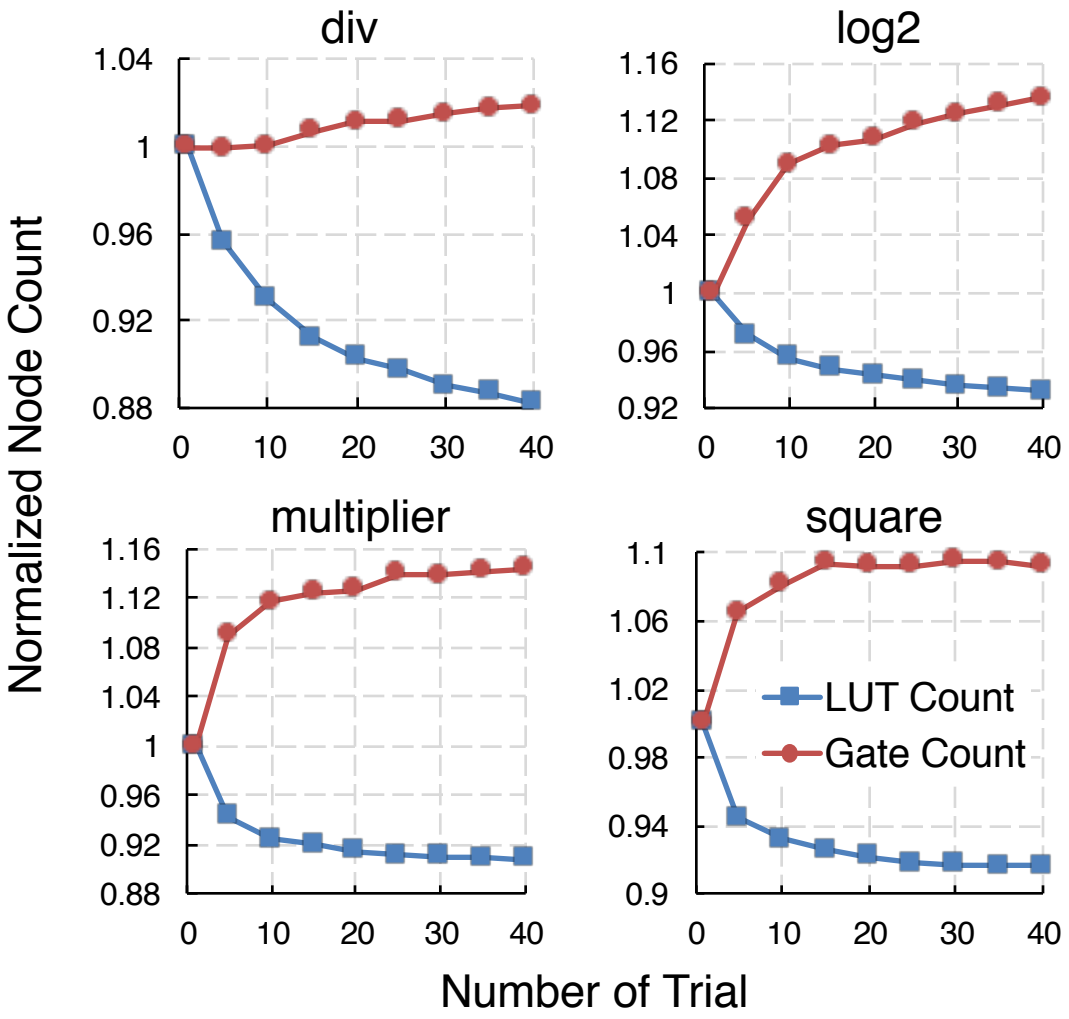


- Initial design: best-known results from EPFL record
- Area improvements
  - EPFL: 7% on average, up to 14%
  - Can effectively handle large circuit (~44k LUTs)
- Also able to improve all 10 control-intensive designs in EPFL benchmark suite

LUT Count vs. Gate Count Reduction

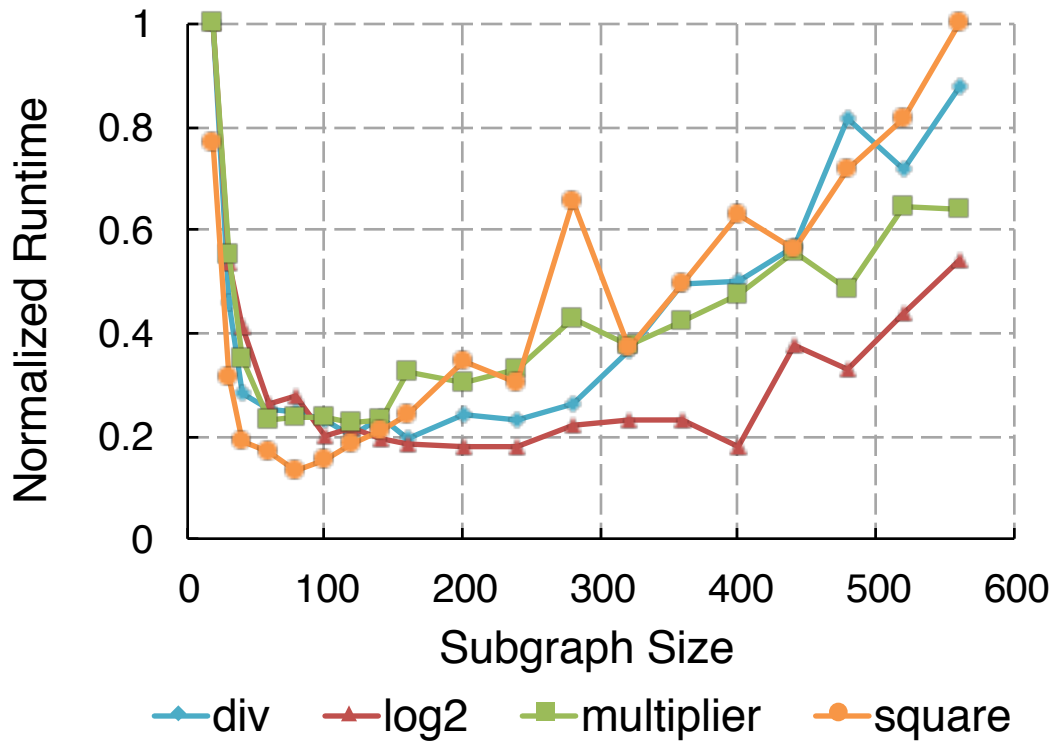
Verified:

post-mapping area **does not necessarily** correlate with **pre-mapping** area



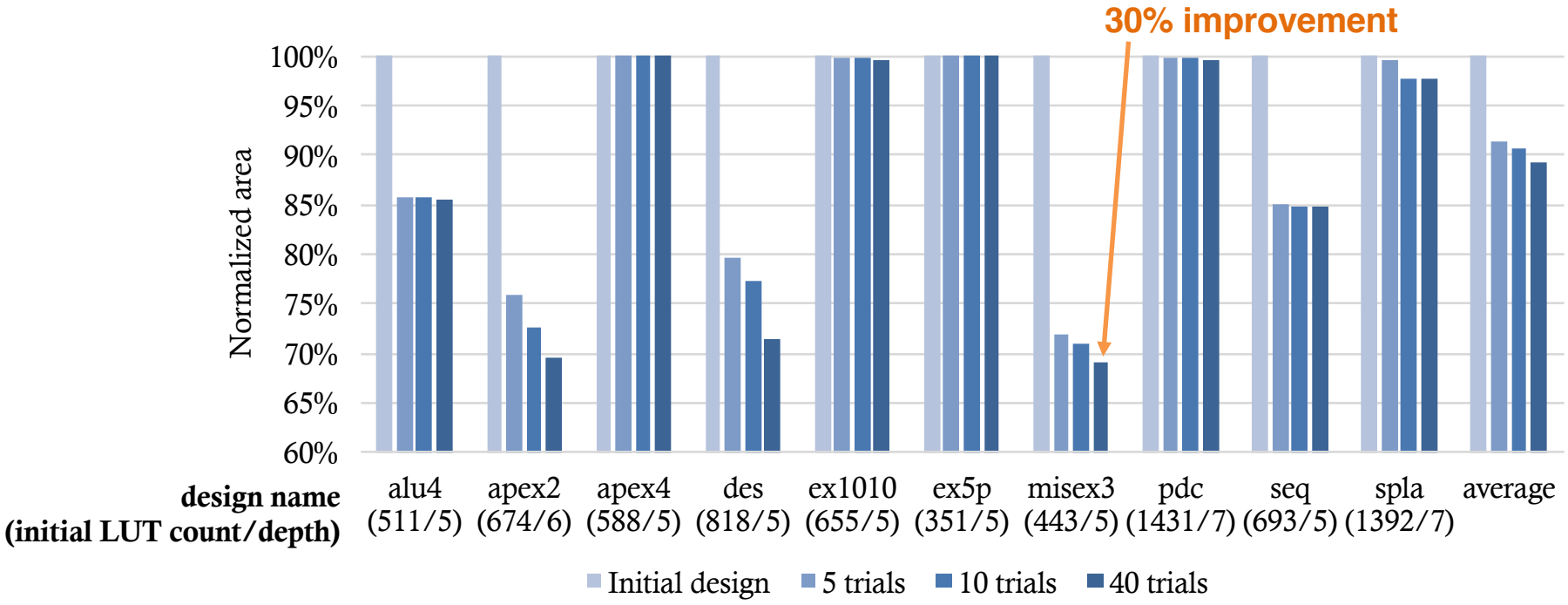
Partition Granularity vs. Runtime

- Trade-off between runtime vs. progress per trial
  - Optimal subgraph size is around 100 LUTs

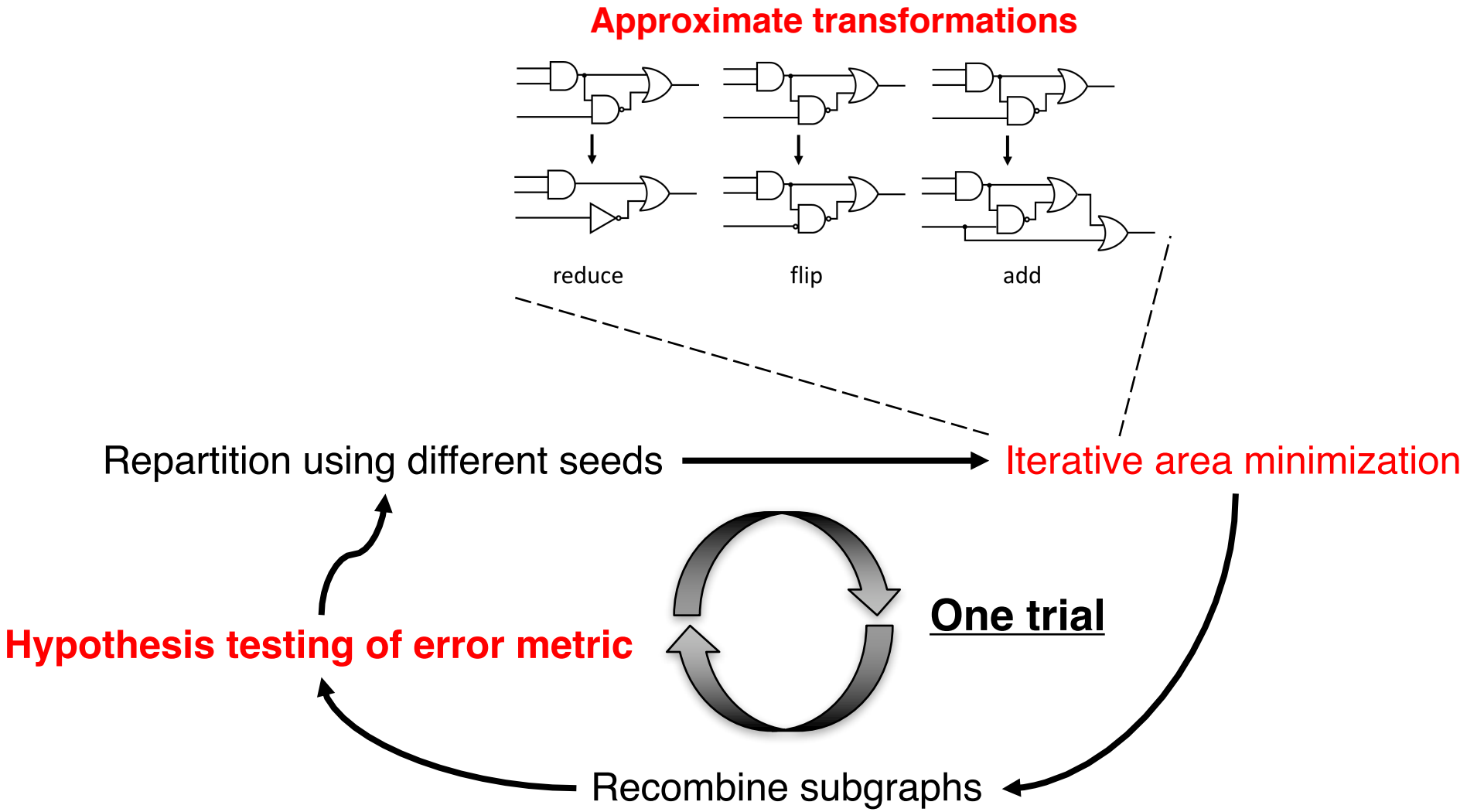


Depth Constrained Area Minimization on MCNC

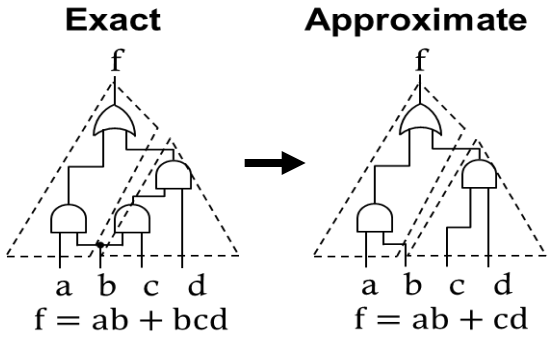
- Constraint: no depth increase compared to initial design
  - Initial designs generated by ABC's depth-minimizing *resyn2* script
- Area improvements under depth constraint for MCNC benchmarks
  - 11% on average, up to 30%



Extending PIMap to Approximate Logic Synthesis



Statistically Certified Approximate Synthesis [ICCAD'17]



- Previous approaches (sample-based testing):
  - Randomly pick N input vectors, then simulate, error rate = $N_{incorrect}/N$

- Our approach: Formally quantify errors using hypothesis testing

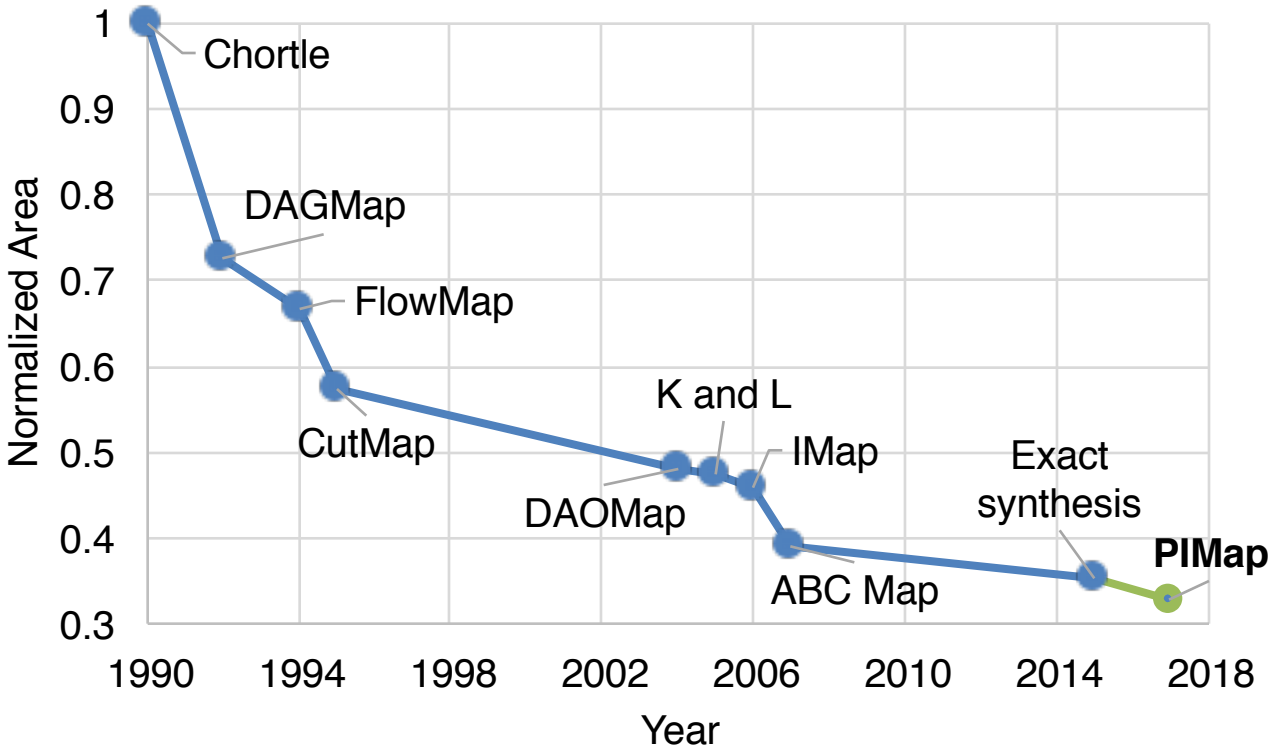
Testing different types of error metrics

Error metric	Test target	Test statistic
Error rate	Sample occurrence	Binomial test
Average error magnitude	Sample mean	T-test
Error variance	Sample variance	χ^2-test

G. Liu and Z. Zhang, **Statistically Certified Approximate Logic Synthesis**, ICCAD'2017.

PIMap Summary

- Current logic synthesis flow still leaves nontrivial room for area improvement (up to 30%)
- Parallelized stochastic optimization is an effective approach for technology-aware synthesis
- Similar opportunities exist in RTL and high-level synthesis



Chortle: Francis, et al., DAC'90
DAGMap: Chen, et al., DT'92
FlowMap: Cong and Ding, TCAD'94
CutMap: Cong and Hwang, FPGA'95
DAOMap: Chen and Cong, ICCAD'04
K and L: Kao and Lai, TDAES'05
Imap: Manohararajah, et al., TCAD'06
ABC Map: Mishchenko, et al., TCAD'07
Exact synthesis: Haaswijk, et al., ASPDAC'17