DREAMPlaceFPGA-PL: An Open-Source GPU-Accelerated Packer-Legalizer for Heterogeneous FPGAs

Rachel Selina Rajarathnam[1], Zixuan Jiang[1], Mahesh A. Iyer[2], David Z. Pan[1]

[1]*ECE Department, The University of Texas at Austin, TX, USA*

[2]*Intel Corporation, CA, USA*

This work was supported in part by Intel and VMware

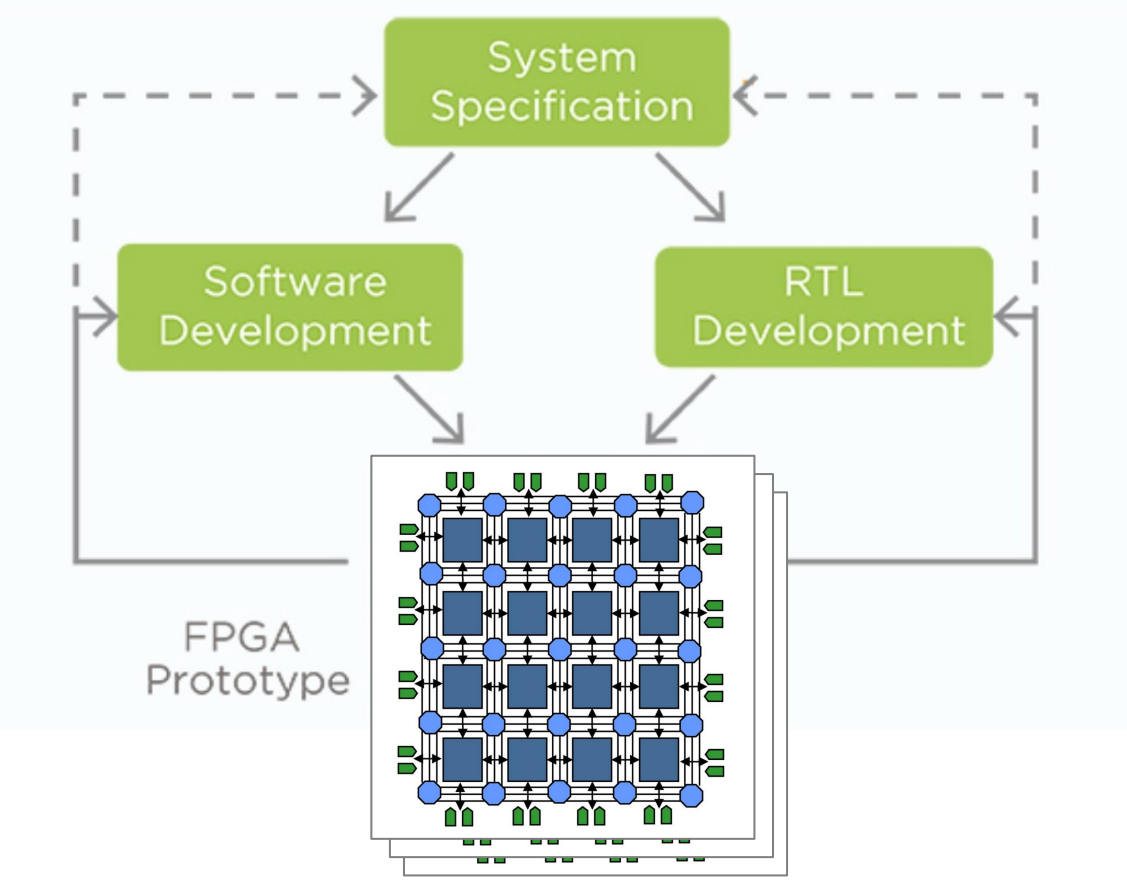
TEXAS
The University of Texas at Austin

intel

OUTLINE

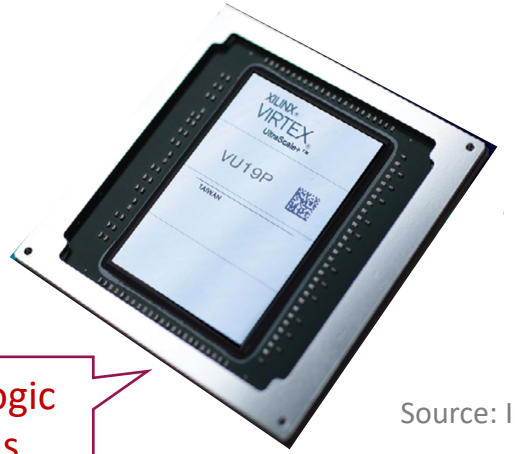
- Background & Motivation
- Previous Works
- DREAMPlaceFPGA-PL
- Results
- Summary

FPGAs in ASIC Prototyping



One or more FPGAs used

10M logic elements



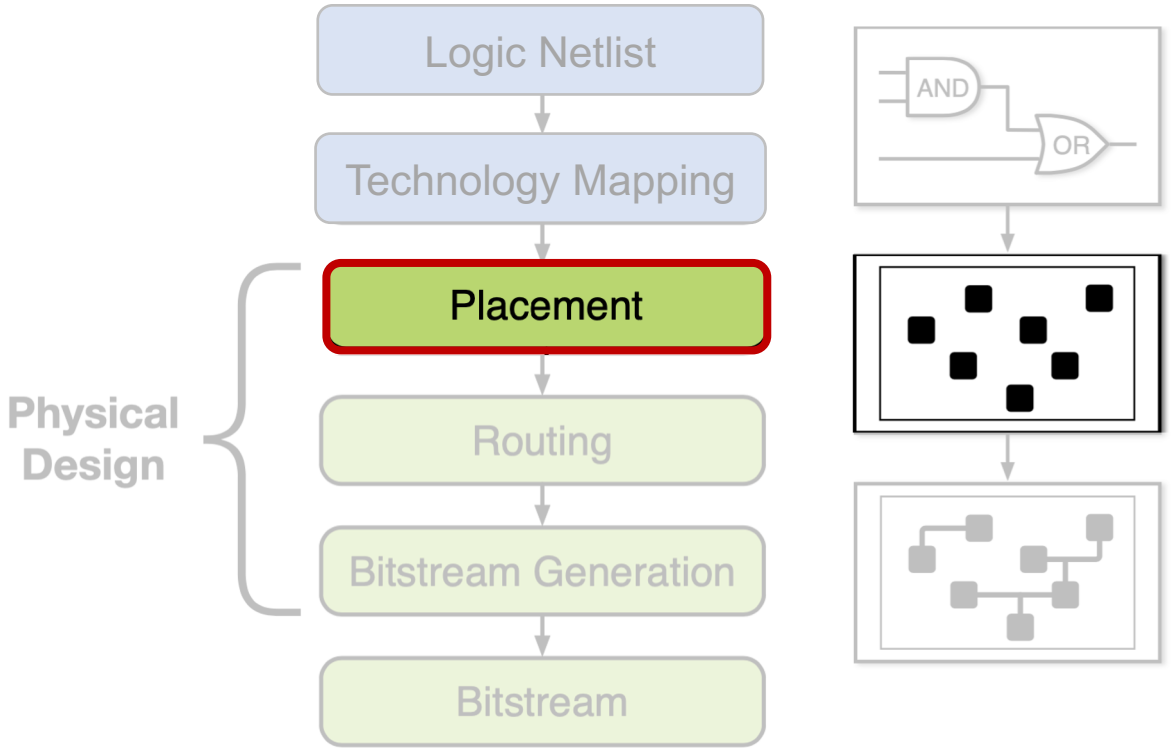
9M logic cells

Source: Intel & AMD

https://sirinsoftware.com/blog/technology/great-challenges-fpga-prototyping-must-overcome/

S. M. Trimberger, *IEEE*, 2015

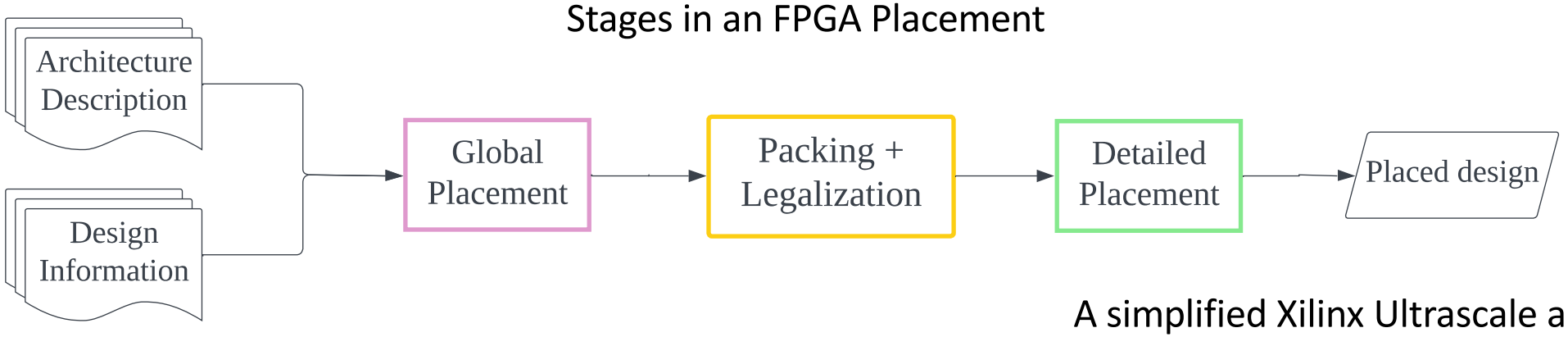
FPGA Design Implementation



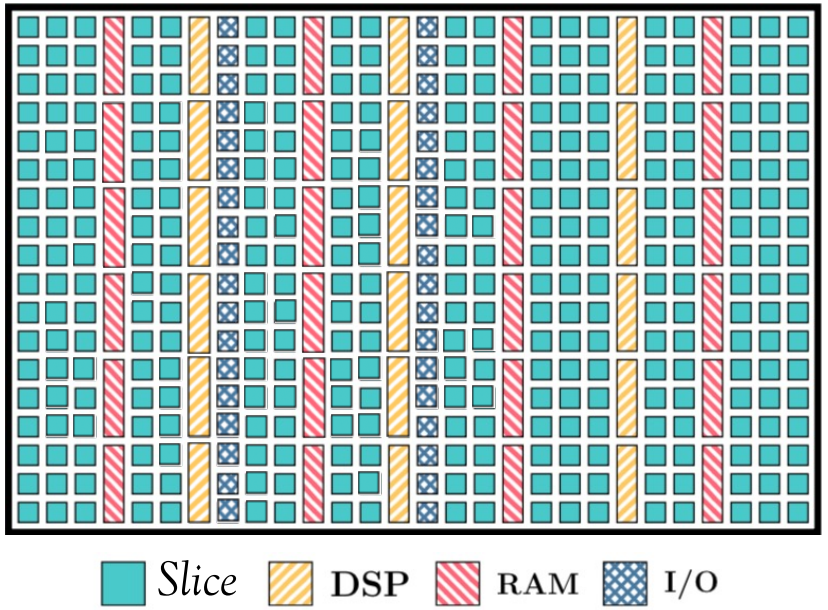
http://wuxili.net/pdf/dissertation.pdf

FPGA PLACEMENT

Stages in an FPGA Placement



A simplified Xilinx Ultrascale architecture



Placement Problem Formulation

$$\min_{x,y} W(x,y) \;\; s.t. \; Constraints$$

Minimize wire length

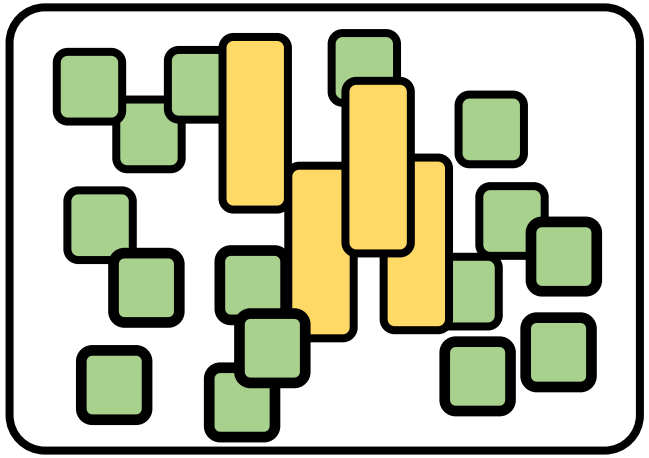
Constraints:

- Density: No overlaps between instances,
- Routability: Design is routable,
- Congestion: Ensure placement is not congested etc.,

Source: Xilinx; ISPD'2016 Contest

Global Placement

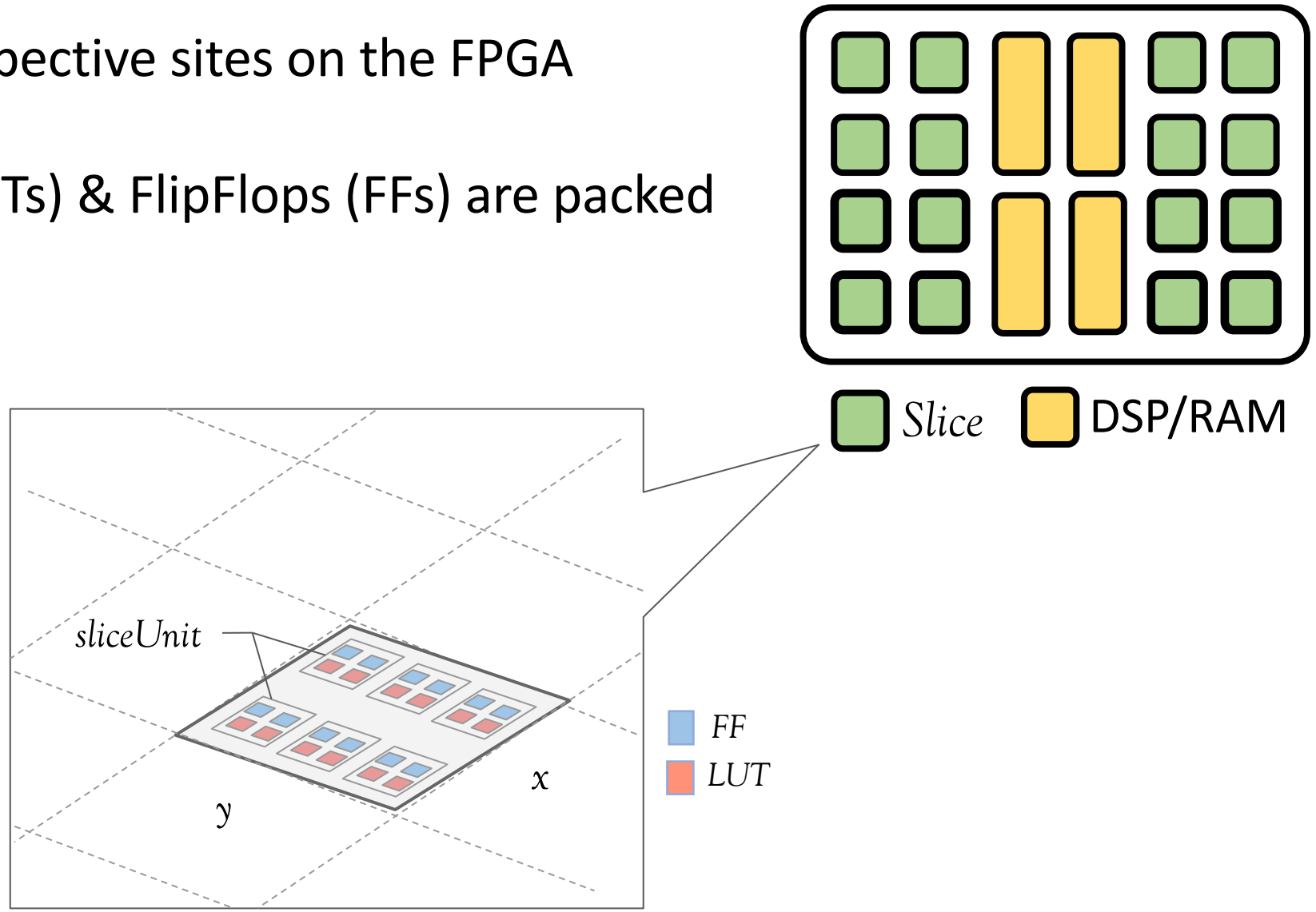
- Obtain rough legal locations for instances
- Optimization problem can be formulated as:
  - Simulated Annealing
  - Quadratic Placement
  - Non-linear Placement



LEGALIZATION

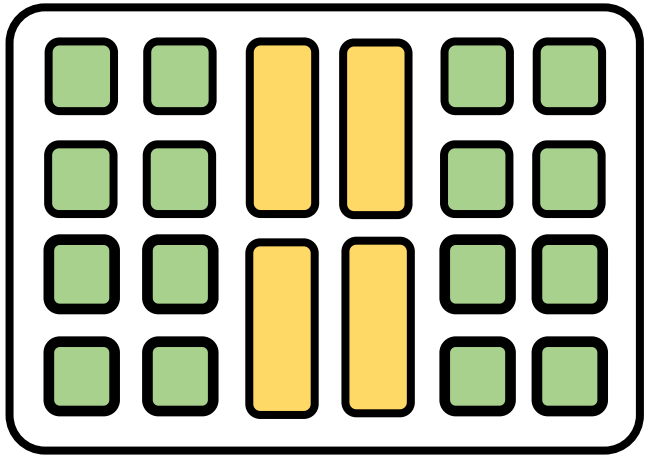
Assign instances to respective sites on the FPGA

- Look-up tables (LUTs) & FlipFlops (FFs) are packed

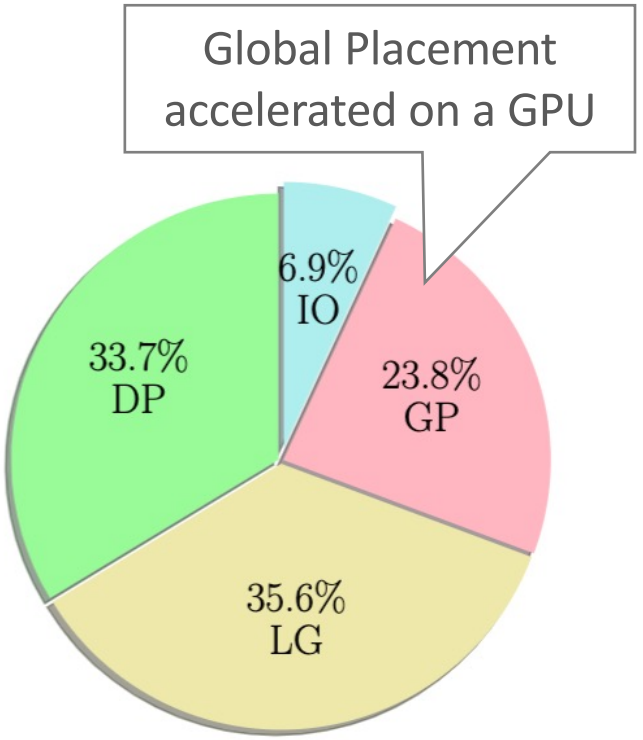


Detailed Placement

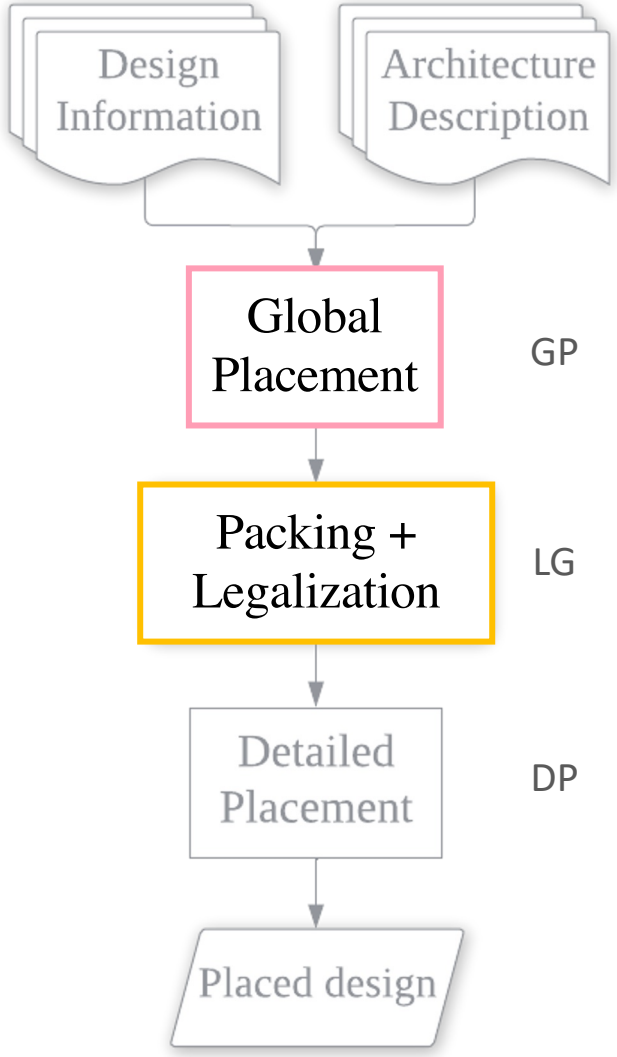
- Further refine the legalized placement
- Formulated as
  - Independent Set Matching
  - Dynamic Programming
  - Hill Climbing, etc.



FPGA Placement Acceleration



DREAMPlaceFPGA Runtime Breakdown



R.S.Rajarathnam+, ASPDAC'2022

EXISTING PACKER-LEGALIZER ACCELERATION SCHEMES

- **Multi-threaded CPU** [W.Li+, ICCAD'2019; W.Li+, TCAD'2017; L.Singhal+, DAC'2017]
  - Scalable consensus-based algorithms

Other acceleration schemes not explored for FPGA packing-legalization!

- Use of **CPU-GPU** systems in **ASIC** LG Acceleration [H.Yang+, DATE'2022]
  - 2x - 4x speedup observed
  - In ASIC, Legalization accounts for ~5% of total placement runtime [Y.Lin+, TCAD'2020]

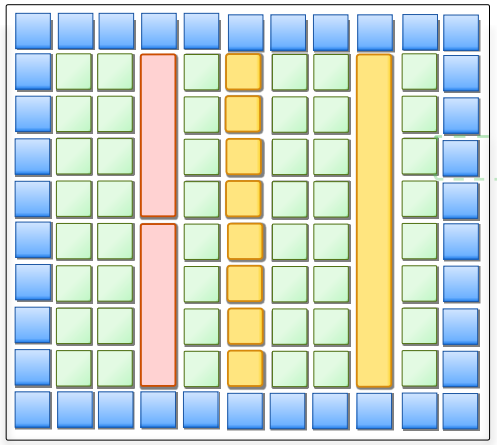
EXISTING PACKER-LEGALIZER ACCELERATION SCHEMES

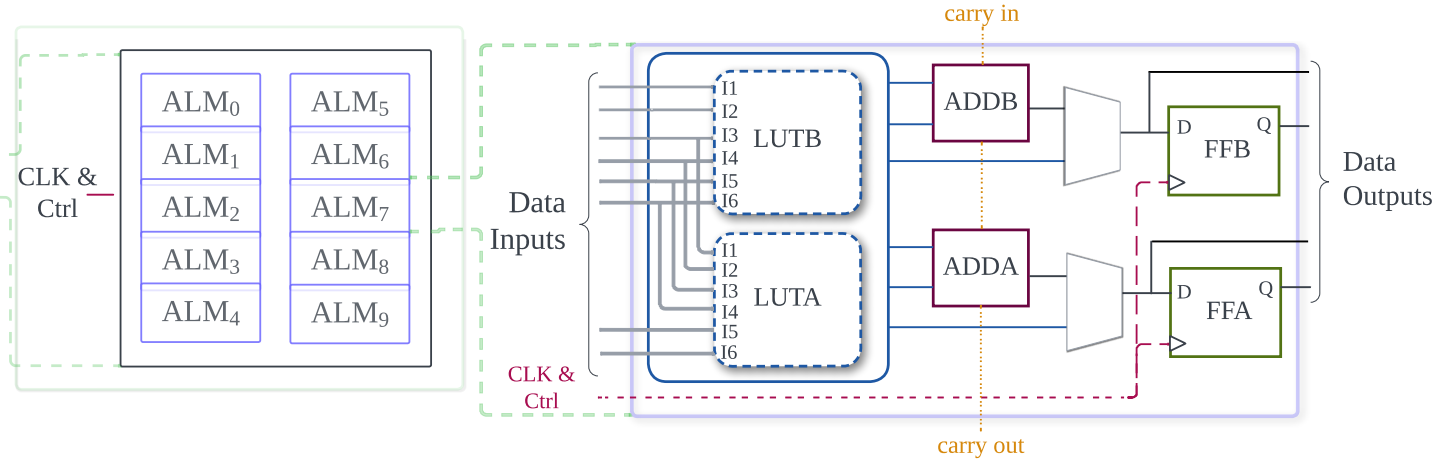
- Multi-threaded CPU [W.Li+, ICCAD'2019; W.Li+, TCAD'2017; L.Singhal+, DAC'2017]
  - Scalable consensus-based algorithms
- Oth
- Us
  - 2x - 4x speedup observed
  - In ASIC, Legalization accounts for ~5% of total placement runtime [Y.Lin+, TCAD'2020]

In FPGAs, LG takes ~35% of total placement runtime

Accelerating on GPUs can help reduce runtime!

PACKING RULES IN FPGA ARCHITECTURES

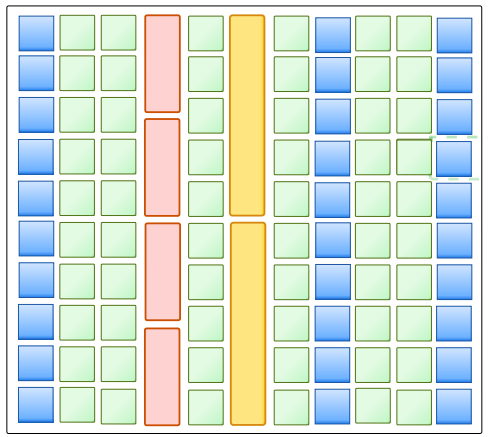


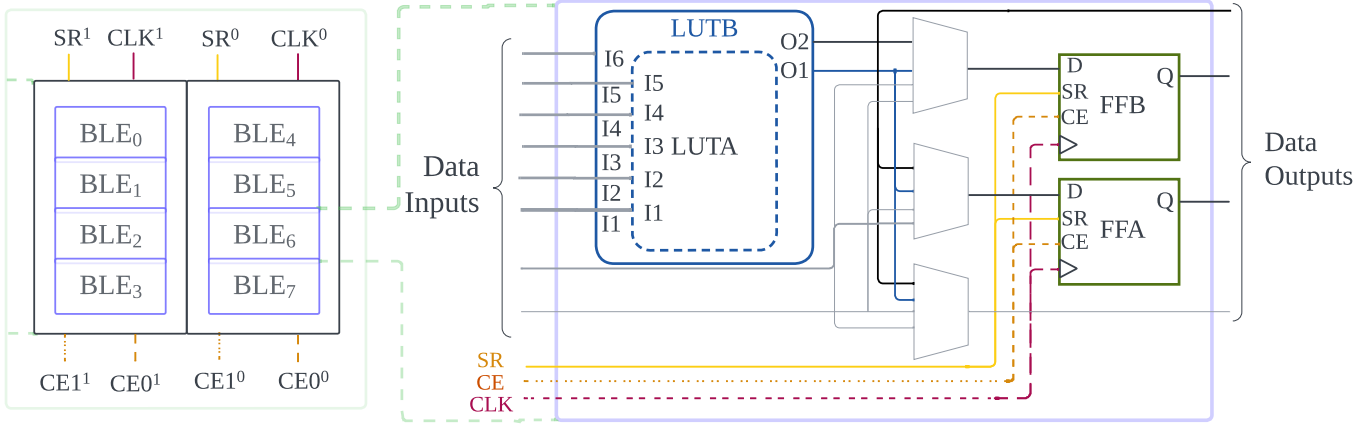


Each Logic Array Block (LAB) contains 10 Adaptive Logic Modules (ALMs).
Each ALM contains 2 LUTs, 2 FFs, and 2 adders.

StratixIV Arch

Slice ⟷ LAB
sliceUnit ⟷ ALM





Each Configurable Logic Block (CLB) contains 8 Basic Logic Elements (BLEs).
Each BLE contains 2 LUTs, and 2 FFs.

Ultrascale Arch

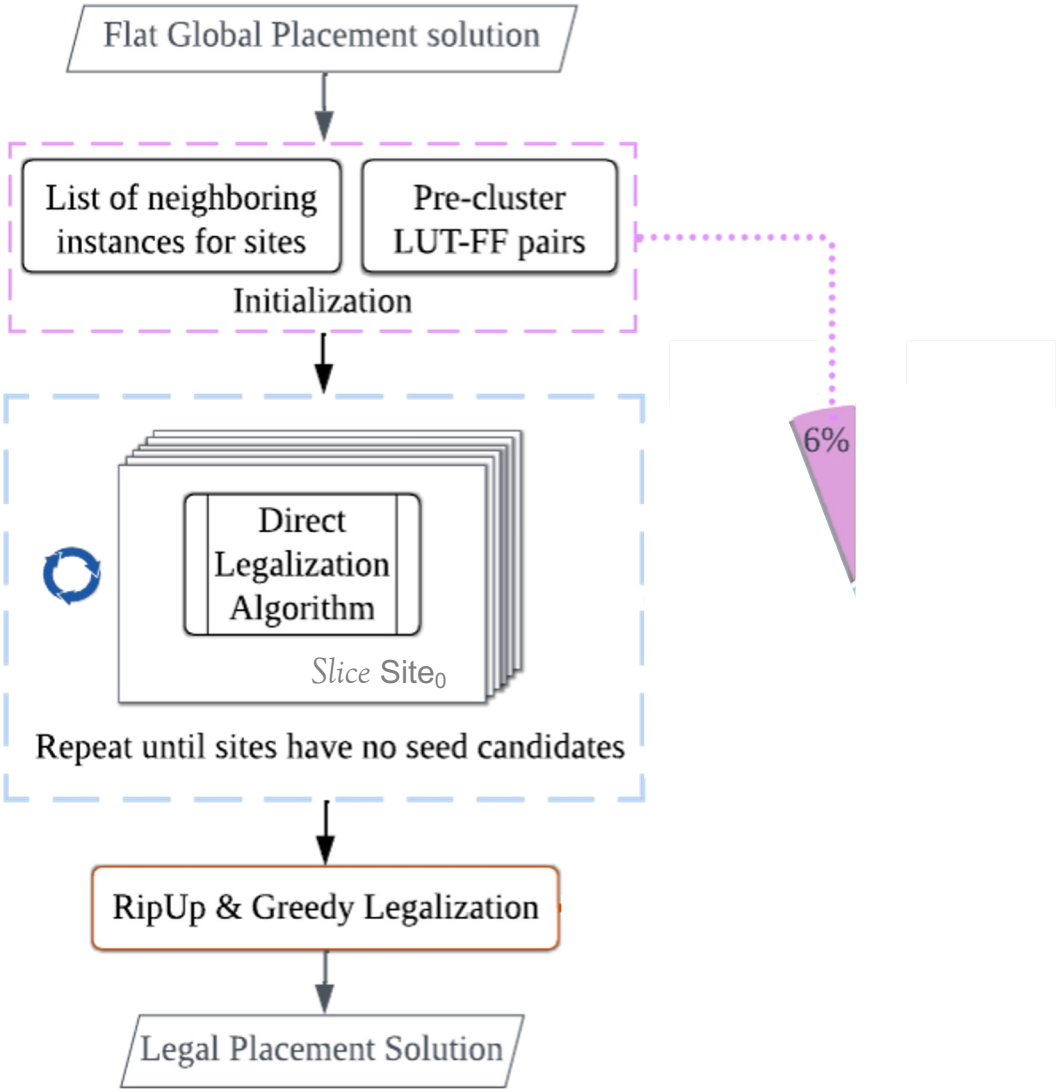
Slice ⟷ CLB
sliceUnit ⟷ BLE

Intel StratixIV: https://www.intel.com/content/www/us/en/products/programmable/devices/features.html; Xilinx Ultrascale: https://www.ispd.cc/contests/16/Legalization.html

SIMULTANEOUS PACK-LEGALIZE ALGORITHM

- Site centric algorithm ⇒ Scalable
- LUTs and FFs in a flat global placement are packed and legalized

- Initialization:
  - Precluster LUT with FFs in fanout
  - Get neighbor instances of each site

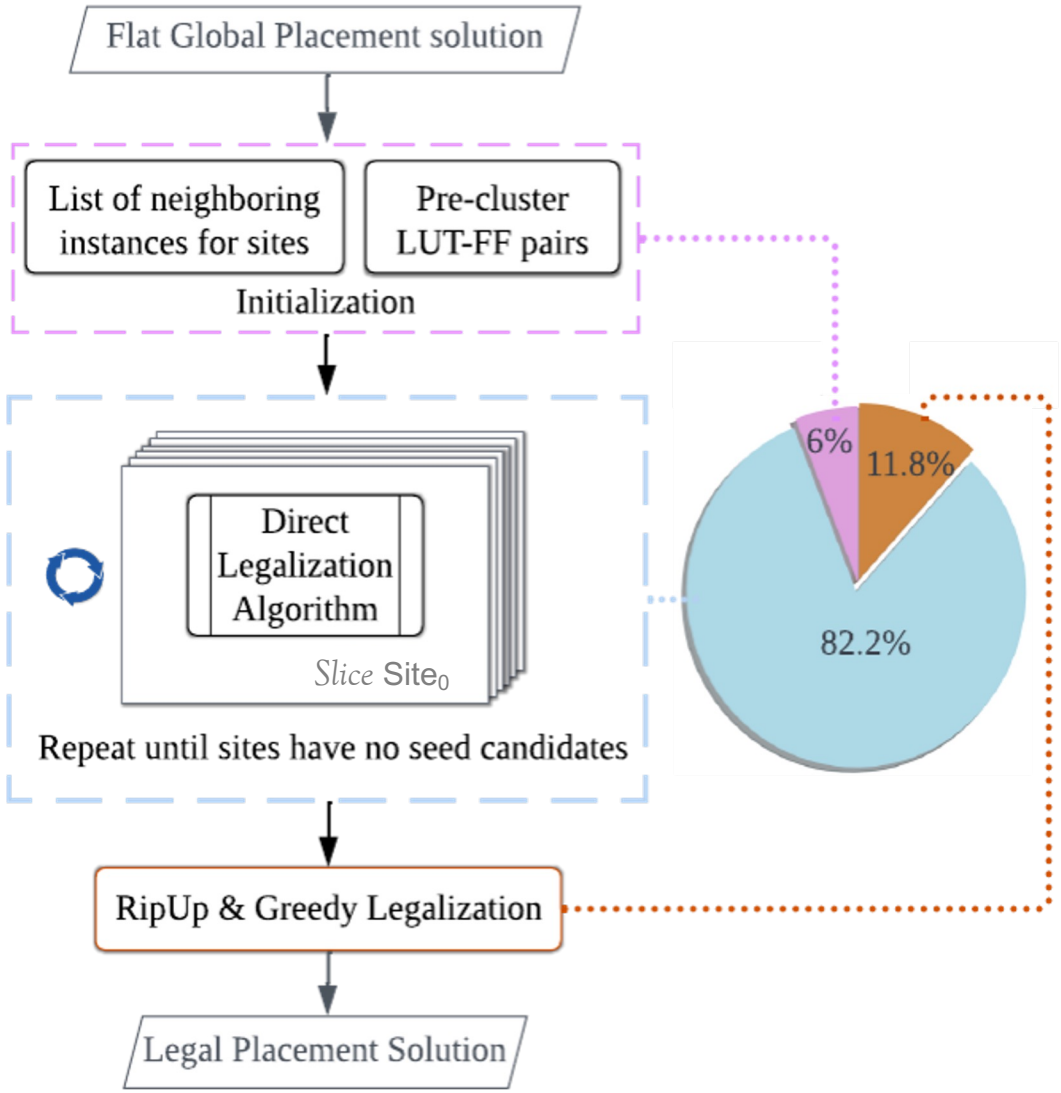


W.Li+, TCAD'2018

SIMULTANEOUS PACK-LEGALIZE ALGORITHM

- Direct Legalization (DL) Algorithm:
  - Best Candidate for GPU Acceleration
  - A Candidate: cluster of LUTs + FFs that can be placed in a *Slice*
  - Each site runs in parallel
  - > 90% LUTs + FFs are legalized
- RipUp & Greedy Legalization:

  Handles the remaining instances



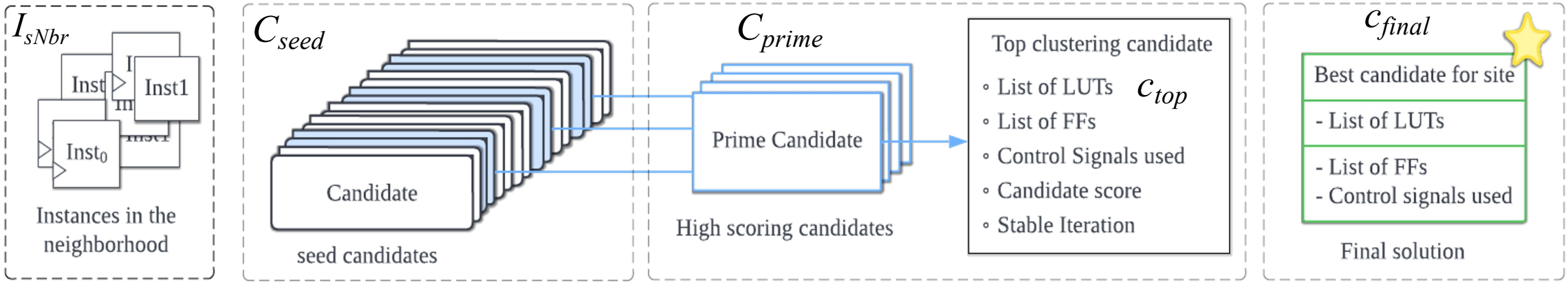
W.Li+, TCAD'2018

DREAMPLACEFPGA-PL

- First attempt to accelerate the pack-legalize stage on GPUs
- Accelerate the Direct Legalization (DL) algorithm
  - Revise runtime-critical portions in the algorithm
  - Employ task scheduling for even load across GPU threads
- Contribute to FPGA open-source ecosystem

W.Li+, TCAD'2018

FLAT DATASTRUCTURES FOR EACH *SLICE*



I_{sNbr} : Set of all the neighboring instances of *Slice* s within a distance D

Nbr : Instances currently considered by the *Slice* - a subset of I_{sNbr}

C_{seed} : Set of cluster candidates for *Slice* s

C_{prime} : Top 10 candidates in C_{seed} based on score (HPWL + internal nets)

c_{top} : Best candidate in C_{prime}

c_{final} : Final solution for *Slice* s

THE DIRECT LEGALIZATION (DL) ALGORITHM

Run in parallel for all *Slice* sites

Algorithm 1 The Direct Legalization Algorithm

```
1:  while |I_sNbr| > 0 or |C_seed| > 0 do
2:      if c_top is stable for T iterations then
3:          c_final = c_top
4:          Clear all seed candidates in C_seed
5:          Add c_top to C_seed as seed candidate
6:      else
7:          Remove invalid candidates in C_seed and C_prime
8:      end if
9:      Remove committed neighbor instances in Nbr and I_sNbr
10:     if |Nbr| < MinNbrCount then
11:         add next group of neighbors from I_sNbr to Nbr;
12:     end if
13:     Create new seed candidates from C_seed and Nbr
14:     Update best site for instances in c_top
15: end while
```

W.Li+, TCAD'2018

THE DIRECT LEGALIZATION (DL) ALGORITHM

Run in parallel for all *Slice* sites

Algorithm 1 The Direct Legalization Algorithm

```
1:  while |I_sNbr| > 0 or |C_seed| > 0 do
2:      if c_top is stable for T iterations then
3:          c_final = c_top
4:          Clear all seed candidates in C_seed
5:          Add c_top to C_seed as seed candidate
6:      else
7:          Remove invalid candidates in C_seed and C_prime
8:      end if
9:      Remove committed neighbor instances in Nbr and I_sNbr
10:     if |Nbr| < MinNbrCount then
11:         add next group of neighbors from I_sNbr to Nbr;
12:     end if
13:     Create new seed candidates from C_seed and Nbr
14:     Update best site for instances in c_top
15: end while
```

A naive GPU Implementation is > 3x slower than 8T CPU

GPU Runtime Bottleneck (RB)!

W.Li+, TCAD'2018

RB1: New Cluster Candidate Creation

- New candidates = Neighbor Instances in Nbr × candidates in C_{seed}
- $Slice$ sites can have different | Nbr | and | C_{seed} | ⇒ Exploration space differs!
  - Some $Slice$ sites can have no new candidates
  - Some could have 100s-1000s of new candidates => Load imbalance!

RB1: Mitigation Schemes

(1) **Staggered addition** of new neighbor instances

- Default: Group of neighbors added
- Stagger $|\varsigma|$: Max of $|\varsigma|$ neighbors added

$|\varsigma|$: Max number of LUTs + FFs in *Slice*

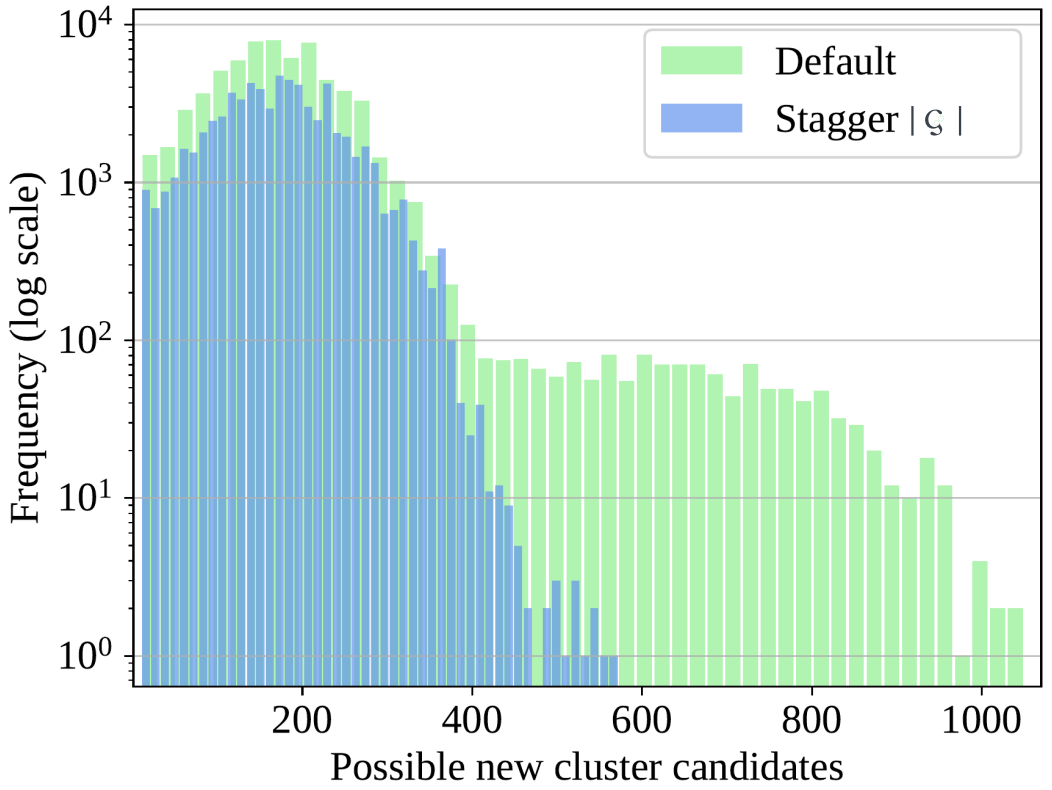
For example, consider a Slice with 100 neighbor instances

All neighbor instances are at a distance 3 from Slice.

Let $|\varsigma| = 32$. *New neighbor addition is as follows:*

Default : {0, 0, 100}

Stagger $|\varsigma|$: {32, 32, 32, 4}



Histogram of new candidate search space of all the sites at the fourth DL algorithm iteration in FPGA12.

RB1: Mitigation Schemes

(1) Staggered addition of new neighbor instances

(2) Scheduling of *Slice* sites

- Order *Slice* sites in decreasing order of new candidate exploration space
- Work on *Slice* sites with similar load at any given time

RB1: Mitigation Schemes

(1) Staggered addition of new neighbor instances

(2) Scheduling of *Slice* sites

(3) Restrict total new candidates explored

- Limit the total | Nbr | x | C_{seed} | combinations explored
- Could degrade placement quality as potential candidates could be discarded

RB2: Instance Best Site Assignment

- Instances in c_{top} select the best $Slice$ site based on:
  - HPWL improvement for instance to move to $Slice$
  - Possiblity of internal nets absorbed in a $Slice$
- Tie-break if multiple $Slice$ sites have same score – low site identifier
- Each Instance selects best site sequentially ⇒ Stalls GPU threads!
- Changes to instance datastructure increases memory footprint significantly

RB2: MITIGATION

- **Minimize sequential portion** and include post-processing step
  - Slower on CPU but much faster on GPU
  - Solution is different but there is no loss of placement quality!

```
set_lock(); //Begin sequential portion
if (site.score == inst.bestScore)
{
    //Tie-break condition
    if (site.id < inst.bestSiteId)
    {
        inst.bestSiteId = site.id;
    }
}
else if (site.score > inst.bestScore)
{
    inst.bestSiteId = site.id;
    inst.bestScore = site.score;
}
unset_lock(); //End sequential portion
```



```
//Sequential assignment of
// instance best score
atomicMax(&inst.bestScore,site.score);

//Parallel post-processing
//Get instance bestSiteId
// based on bestScore
for (    sites in neighborhood of inst)
{
    if (site.score == inst.bestScore &&
        site.top.candidate contains inst &&
        site.id < inst.bestSiteId) //tie-break
    {
        inst.bestSiteId = site.id;
    }
}
//End of parallel post-processing
```

EXPERIMENTAL SETUP

- Design suite: ISPD'2016 benchmarks
- Placers: `elfPlace`
- Router: Xilinx Vivado v2015.4
- Comparison Metrics
  - Placement HPWL, Routed Wirelength
  - Placement Runtime

Design	#IO	#RAM	#DSP	#LUT	#FF
FPGA01	156	0	0	50k	55k
FPGA02	156	100	100	100k	66k
FPGA03	406	600	500	250k	170k
FPGA04	406	600	500	250k	172k
FPGA05	406	600	500	250k	174k
FPGA06	606	1000	600	350k	352k
FPGA07	606	1000	600	350k	355k
FPGA08	406	600	500	500k	216k
FPGA09	606	1000	600	500k	366k
FPGA10	606	1000	600	350k	600k
FPGA11	606	1000	400	480k	363k
FPGA12	406	600	500	500k	602k
Resources	2580	1728	768	538k	1075k

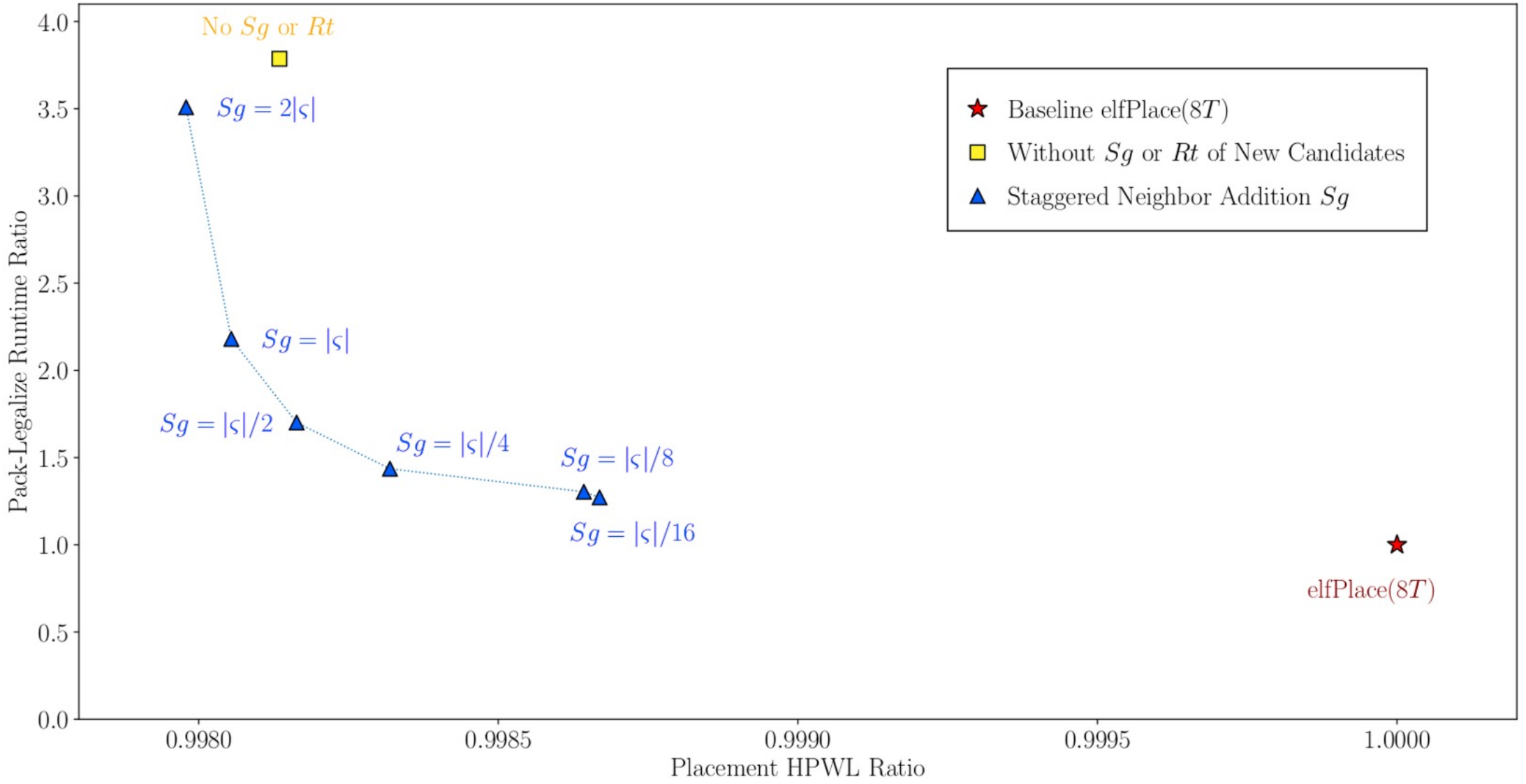
*DP run on `elfPlace`

W. Li+, ICCAD'2019; S. Yang+, ISPD'2016

STAGGERED NEIGHBOR ADDITION

CPU: Intel Core i9-7900X @ 3.30 GHz

GPU: NVIDIA TITAN Xp (Pascal)



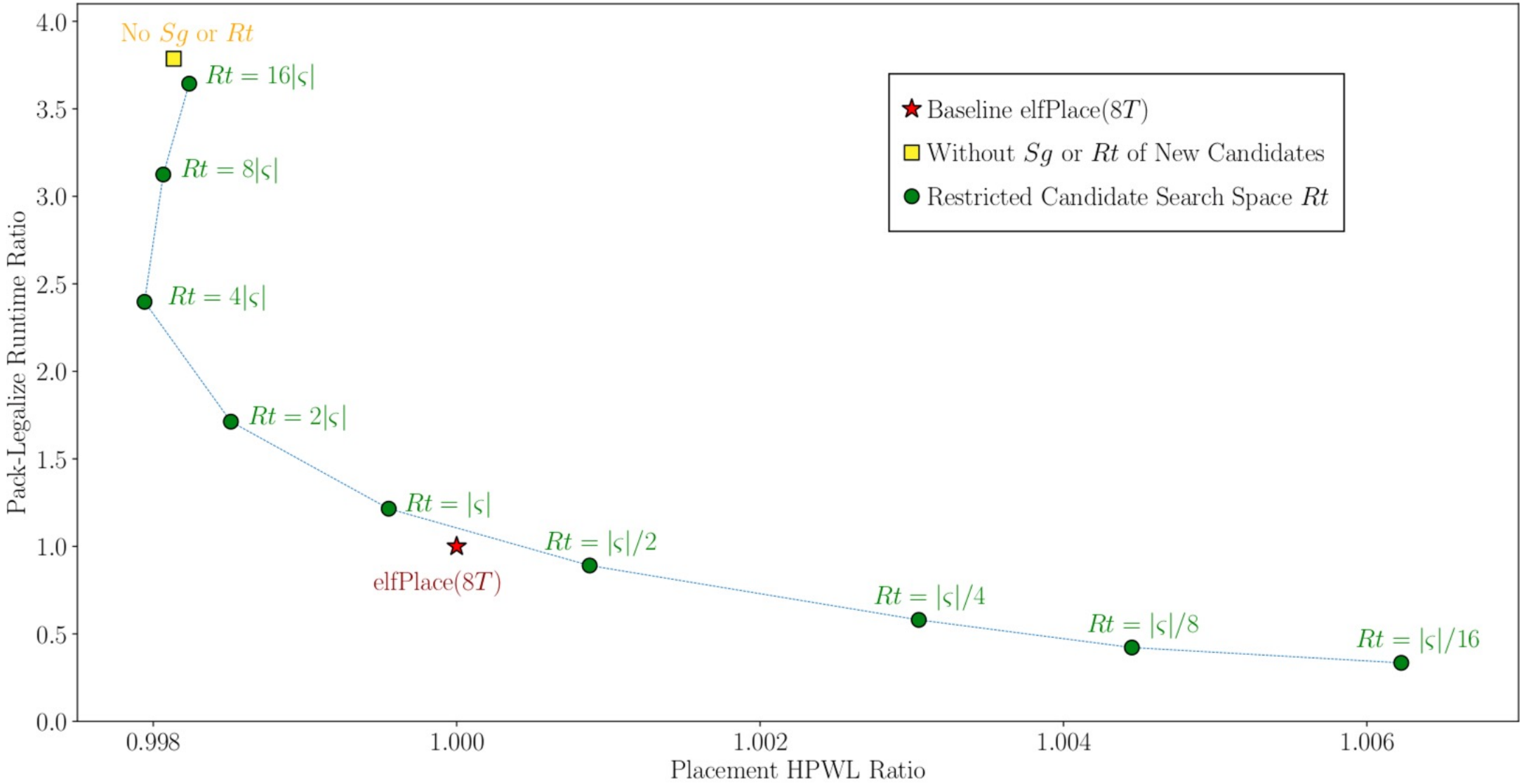
$|\varsigma| = 32$

W. Li+, ICCAD'2019

RESTRICTED NEW CANDIDATE EXPLORATION

CPU: Intel Core i9-7900X @ 3.30 GHz

GPU: NVIDIA TITAN Xp (Pascal)



$|\varsigma| = 32$

W. Li+, ICCAD'2019

SG + RT VARIANTS

Two variants that produce the best runtime vs QoR tradeoff considered are:

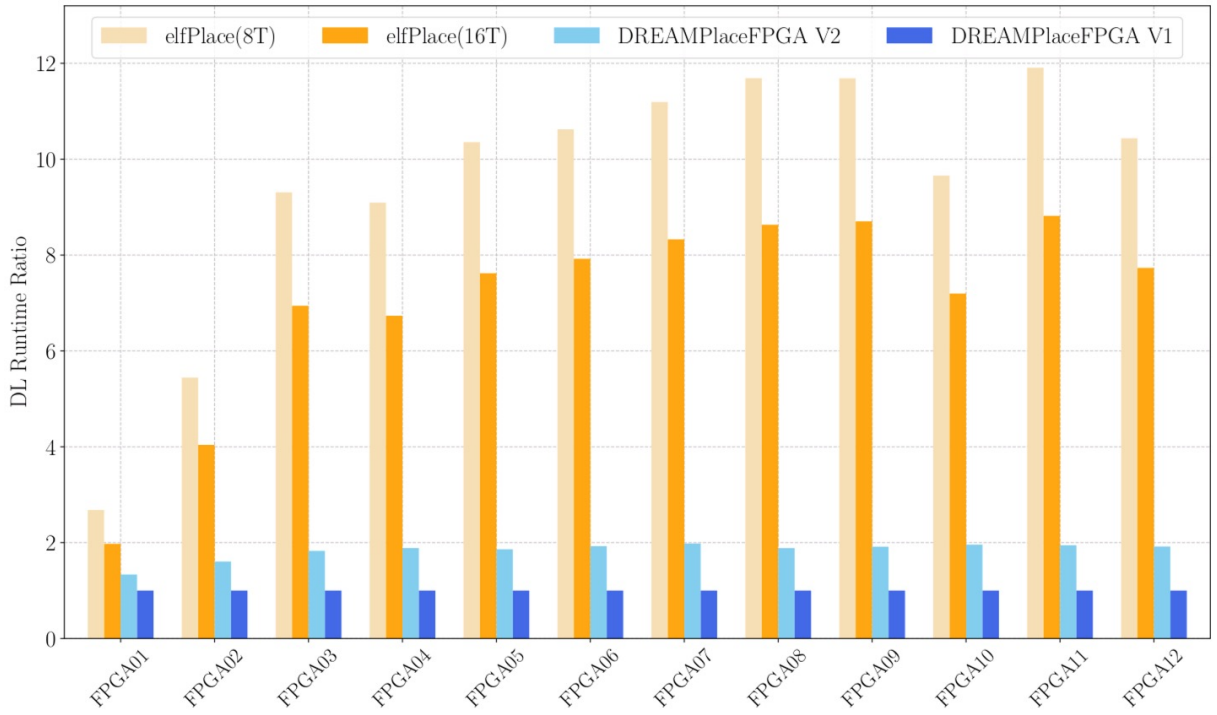
- $V1: Sg = |\varsigma|/16 + Rt = |\varsigma|/8$
  - Aggressive limits on Staggered neighbor addition and Restricted search space
  - Achieves the **best runtime speedup** with a slight degradation of QoR
- $V2: Sg = |\varsigma|/8 + Rt = |\varsigma|/4$
  - Less aggressive than V1
  - Has lower QoR degradation and runtime improvement

RUNTIME IMPROVEMENTS

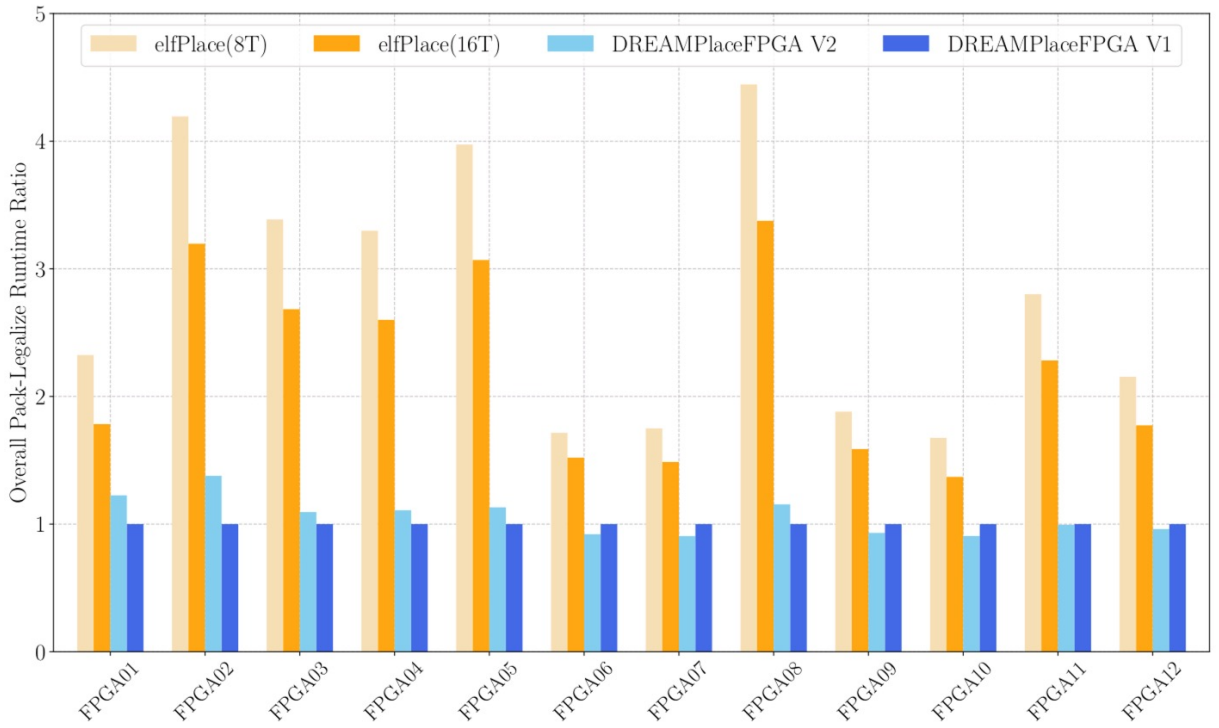
CPU: Intel Core i9-7900X @ 3.30 GHz

GPU: NVIDIA TITAN Xp (Pascal)

Direct Legalization (DL) Runtime



Pack-Legalize (LG) Runtime

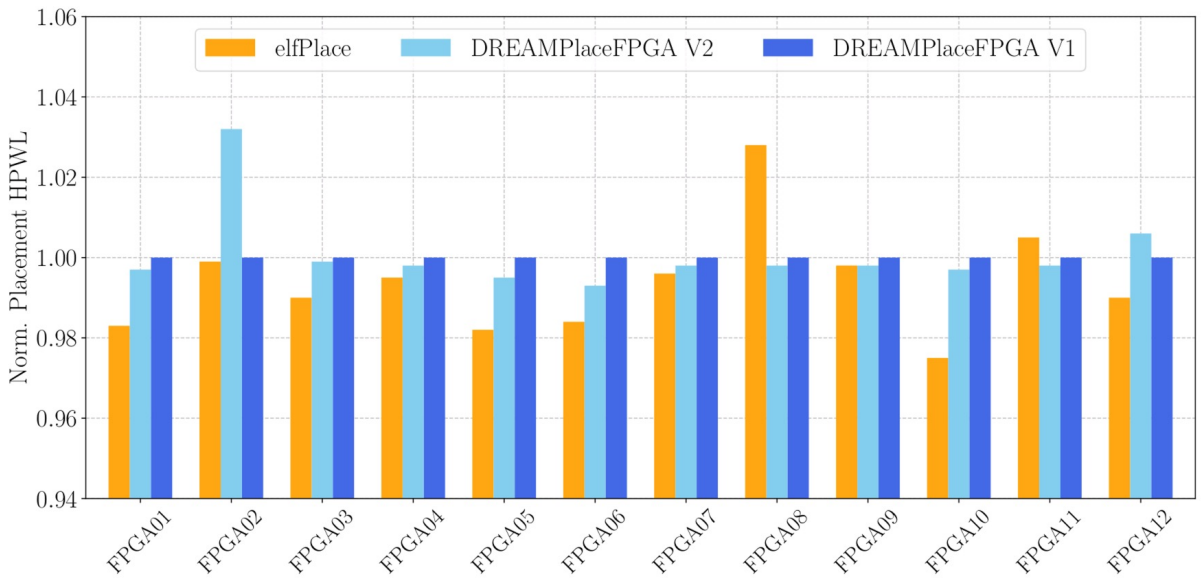


Metric	elfPlace(8T)	elfPlace(16T)	Ours V2	Ours V1
Init	0.97	0.97	**0.92**	1.00
Direct Legalization (DL)	9.51	7.05	1.84	**1.00**
RipUp & Greedy LG (RG)	0.79	0.76	**0.74**	1.00
Total LG	2.80	2.23	1.06	**1.00**

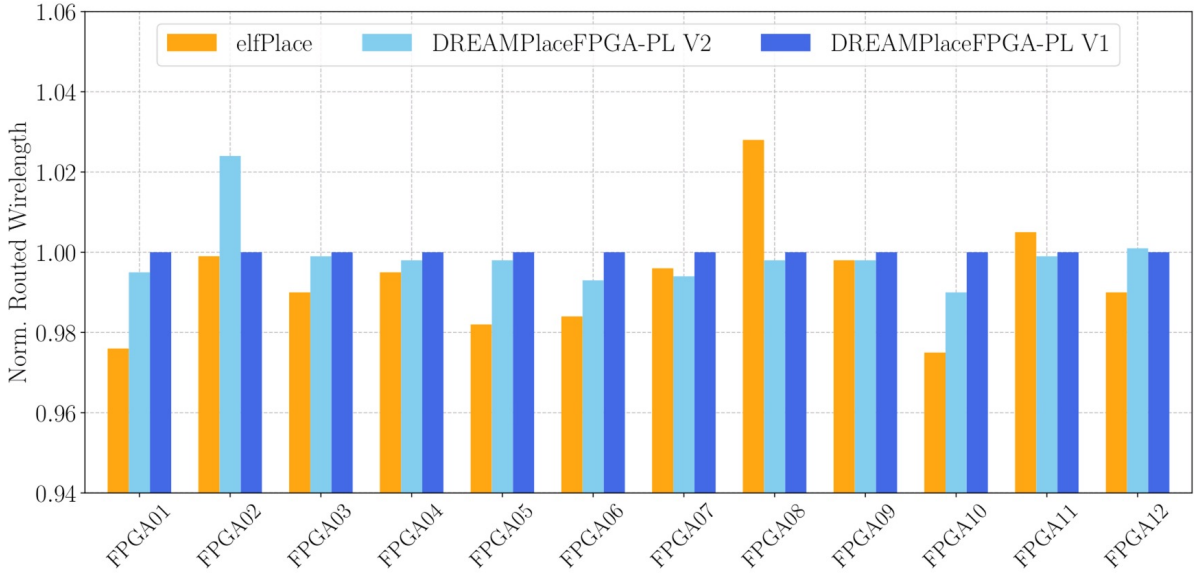
W. Li+, ICCAD'2019

IMPACT ON PLACEMENT QUALITY

Placement HPWL



Routed Wirelength



Metric	elfPlace	Ours V2	Ours V1
Placement HPWL	0.994	0.999	1.000
Routed Wirelength	0.991	0.997	1.000

W. Li+, ICCAD'2019

DREAMPLACEFPGA-PL SUMMARY

- First attempt to accelerate FPGA packer-legalizer on GPU
- Revise the direct legalization algorithm => **7x** speedup than 16T CPU
- `DREAMPlaceFPGA-PL`
  - **2.23x** faster LG than 16T CPU
  - +0.6% placement HPWL and +0.9% routed WL
- With GP and LG accelerated on GPU, placement is **2.38x** faster than 16T CPU

https://github.com/rachelselinar/DREAMPlaceFPGA

THANK YOU