RapidLayout:

Fast Hard Block Placement of FPGA-optimized Systolic Arrays using Evolutionary Algorithms

Niansong Zhang, Xiang Chen, Nachiket Kapre

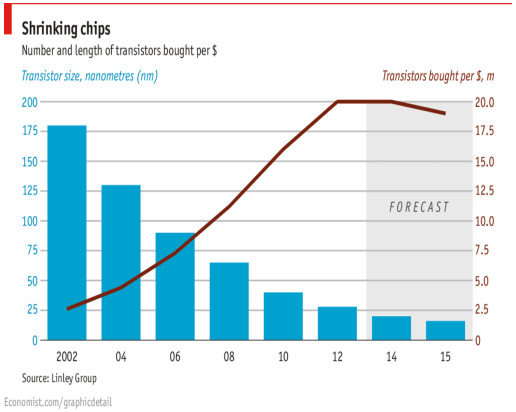
May 24, 2020

Sun Yat-sen University, The University of Waterloo

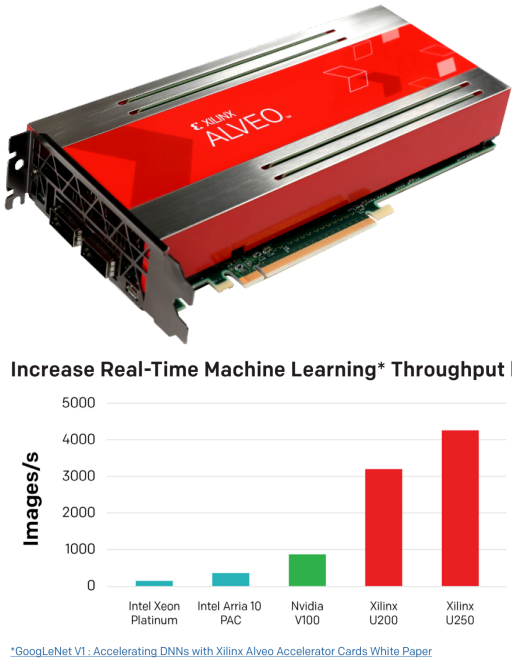
Table of Contents

Backgrounds

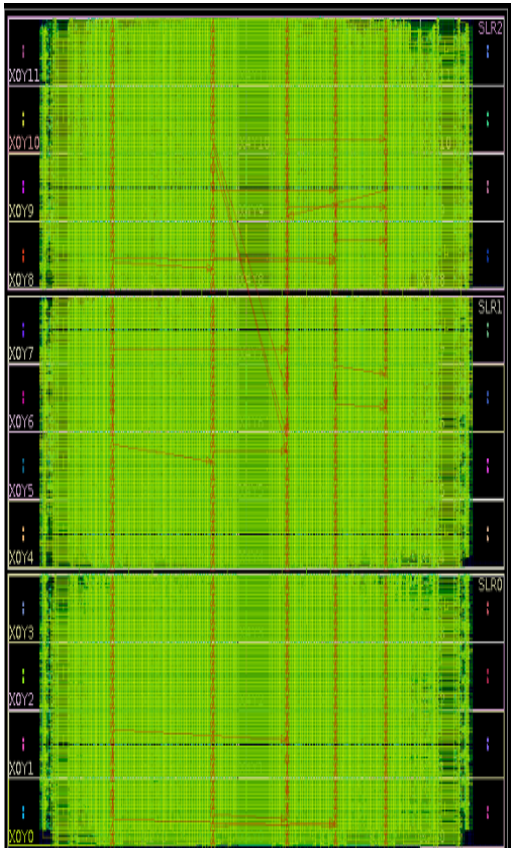
Backgrounds



(a) "Moore's Law" is ending



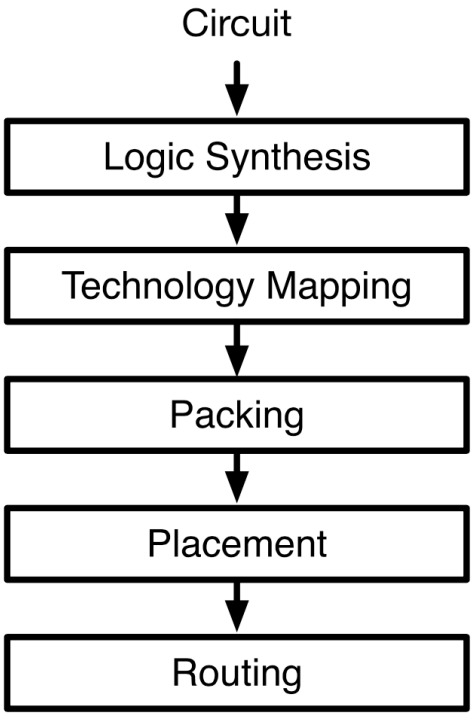
(b) Xilinx ALVEO FPGA



(c) Vivado fails routing

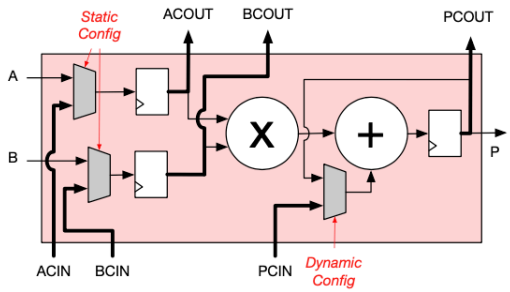
Figure 1: Current EDA tools are challenged by large high-density designs

FPGA Implementation and Placement Algorithms

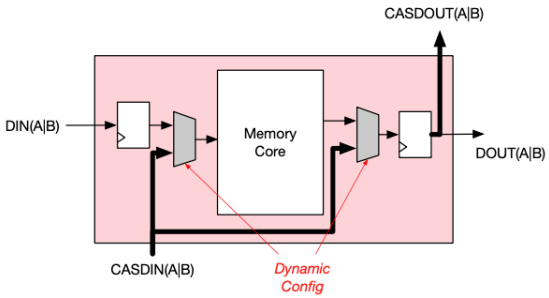


1. Simulated Annealing
   - Near optimal results
   - Not scalable, hard to parallelize
2. Min-Cut
   - Short runtime
   - Likely to stuck in local optima
3. Analytic Placement
   - State-of-the-Art
   - Complex legalization
4. Evolutionary algorithms
   - Inferior to SA
   - Friendly to parallelism

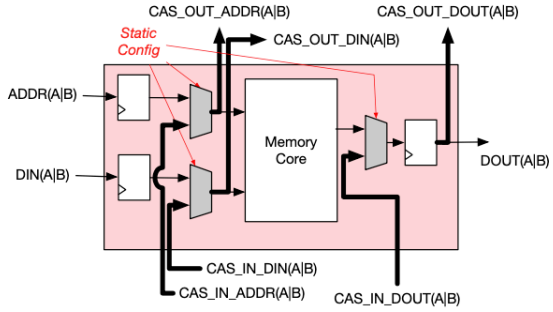
RapidWright and Hard Blocks



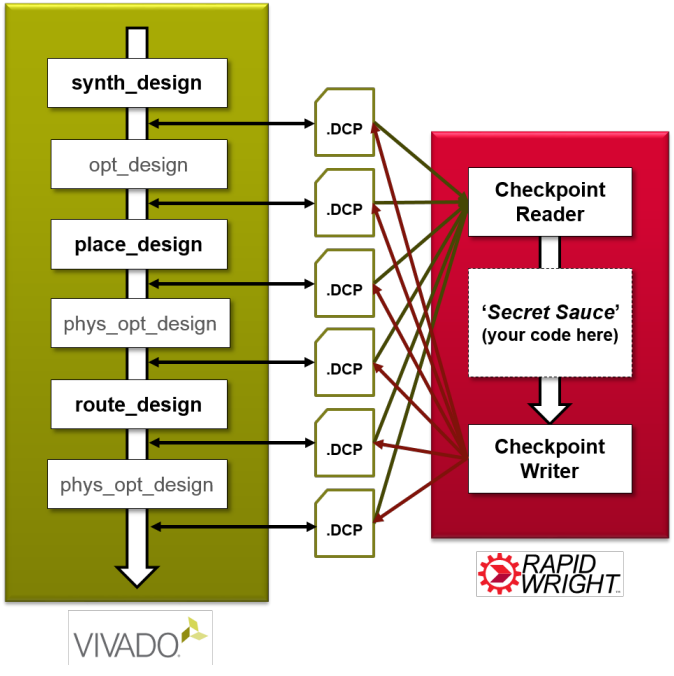
(a) DSP48 cascade (891 MHz)



(b) RAMB18 cascade (825 MHz))



(c) URAM288 cascade (650 MHz)



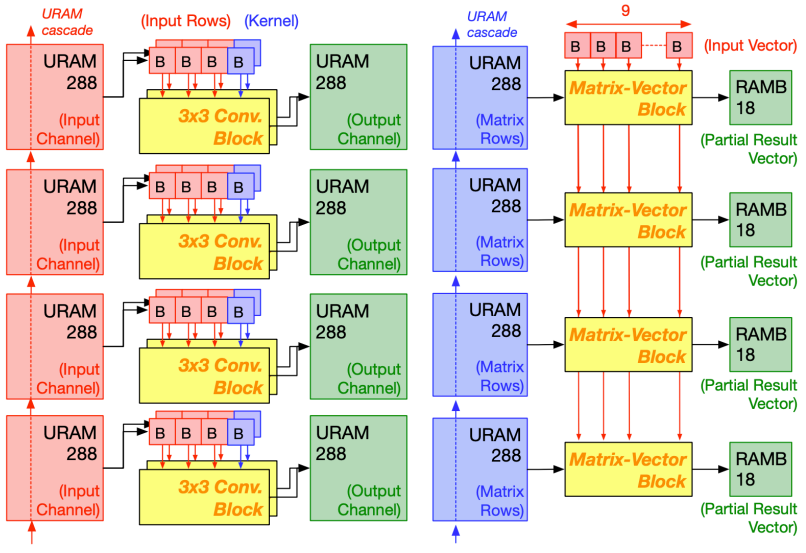
UltraScale+ Hard Blocks

- **DSP48E2:** 27×18 Mult 48-bit Add
- **RAMB18:** 18KB TDP/SDP, supporting cascade
- **URAM288:** 288KB, supporting cascade

RapidWright FPGA framework

- Read/Write DCP files
- Modular design methods
- Logic/Physical netlist modification
- Fast access to physical information

Systolic Arrays and Evolutionary Algorithms



(a) 3×3 Convolution Tiles (b) Matrix-Vector Multiply Tiles

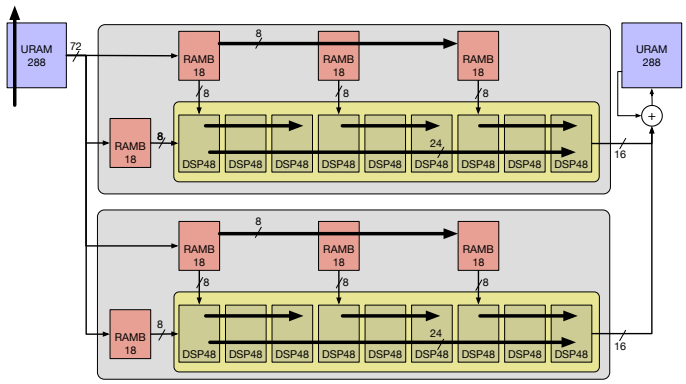
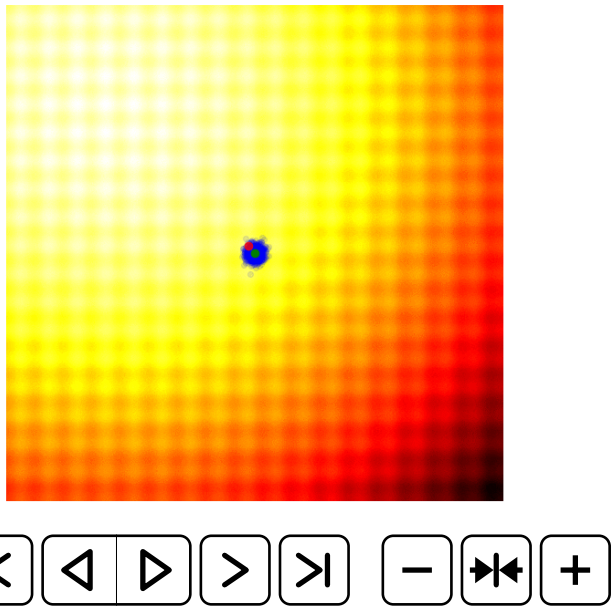


Figure 2: Our systolic array design



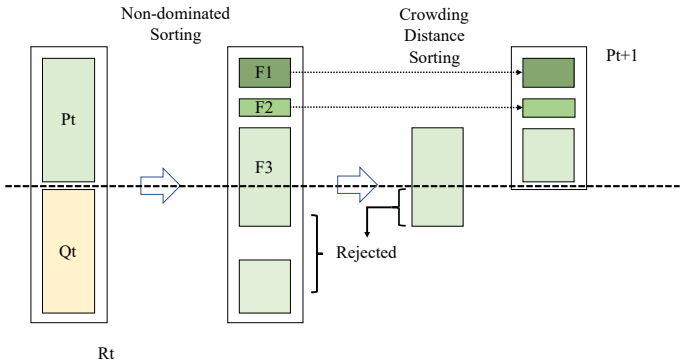
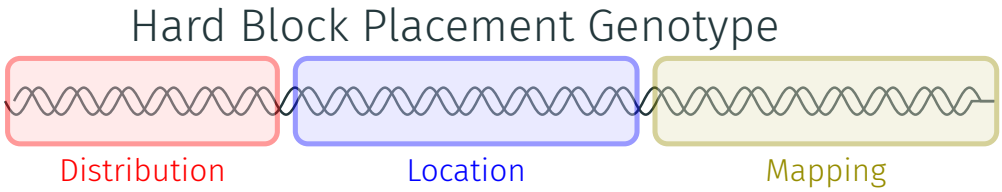
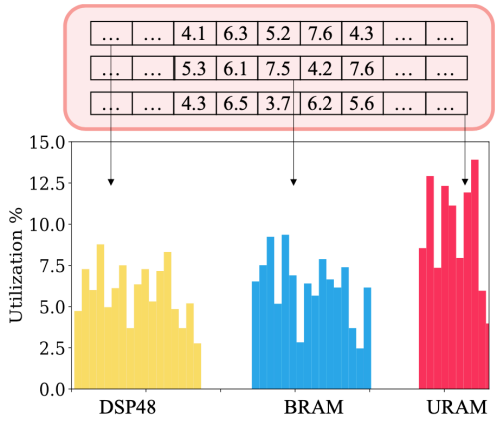


Figure 3: CMA-ES and NSGA-II

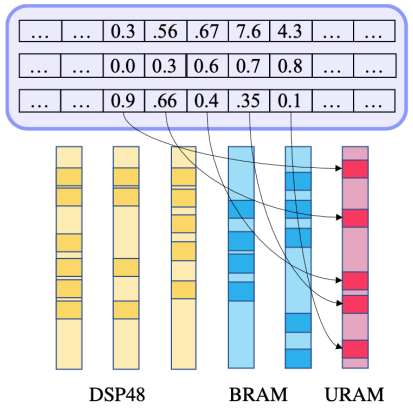
Evolutionary Hard Block Placement

Problem Formulation

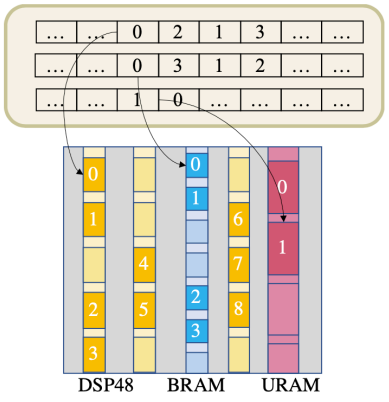




(a) distribution



(b) location



(c) mapping

Minimize:

$$\sum_{i,j}((\Delta x_{i,j} + \Delta y_{i,j}) \cdot w_{i,j})^2 \quad (1)$$

$$\max_k(BBoxSize(C_k)) \quad (2)$$

subject to:

$$0 \leq x_i < XMAX,\ 0 \leq y_i < YMAX \quad (3)$$

$$x_i, y_i \neq x_j, y_j \quad (4)$$

if block i is cascaded after j : $x_i = x_j$

$$y_i = \begin{cases} y_j + 1 & i,j \in \{DSP, URAM\} \\ y_j + 2 & i,j \in \{RAMB\} \end{cases} \quad (5)$$

RapidLayout Workflow

A Netlist Replication:
Takes one computation unit's logic netlist as input and finds the minimum replicating rectangle and replicate the logic netlist.

B Evolutionary Placement:
Solves the placement problem with NSGA-II or CMA-ES.

C Placement and Site Routing:
Embedsthe placement result into DCP checkpoint.

D Post-Placement Pipelining:
Determines pipeline depth based on wirelength.

E SLR Placement and Routing:
Calls Vivado to finish routing of 1 SLR.

F SLR Replication:
Replicates the routed design to fill the FPGA.

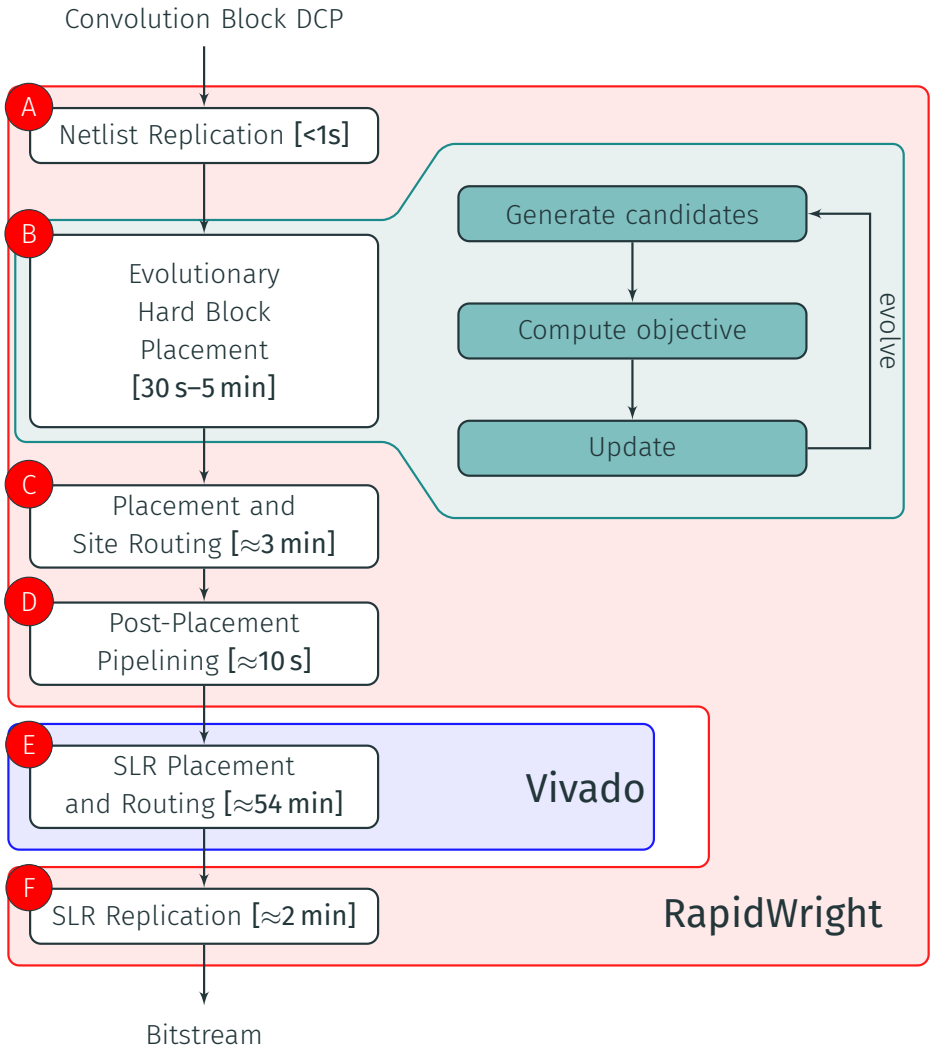


Figure 4: RapidLayout Workflow

Example: Xilinx UltraScale+ VU11P

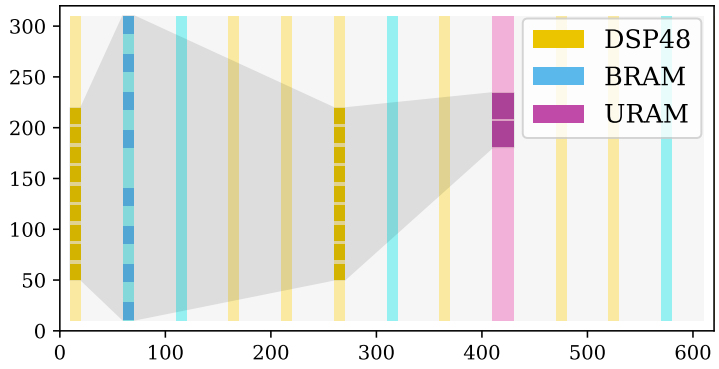


Figure 5: 1 convlution unit

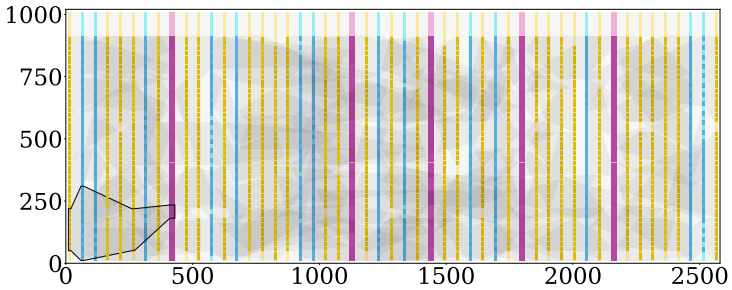


Figure 6: Minimum Replicating Rect

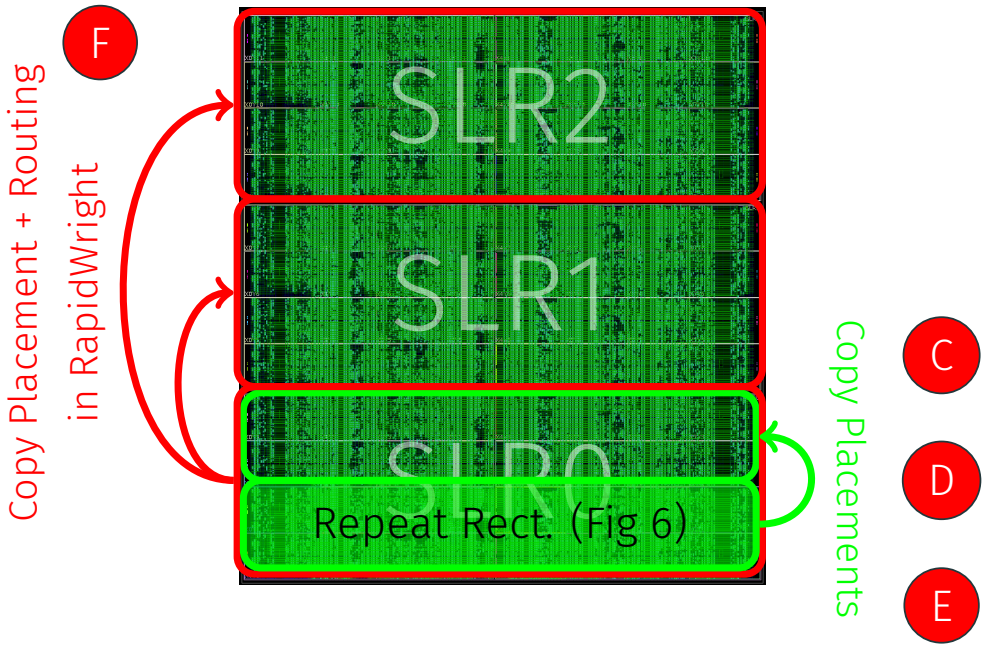


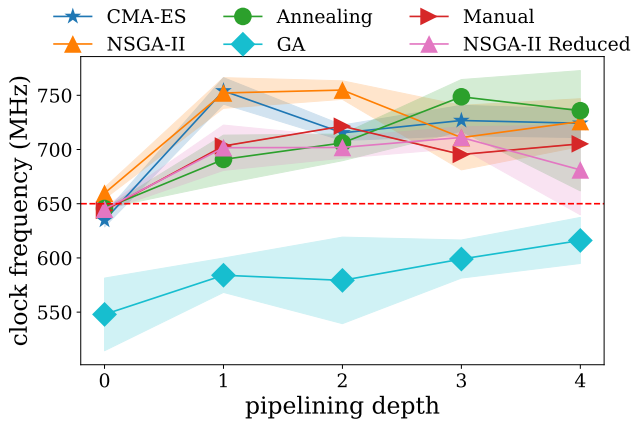
Figure 7: Placement & SLR re-use

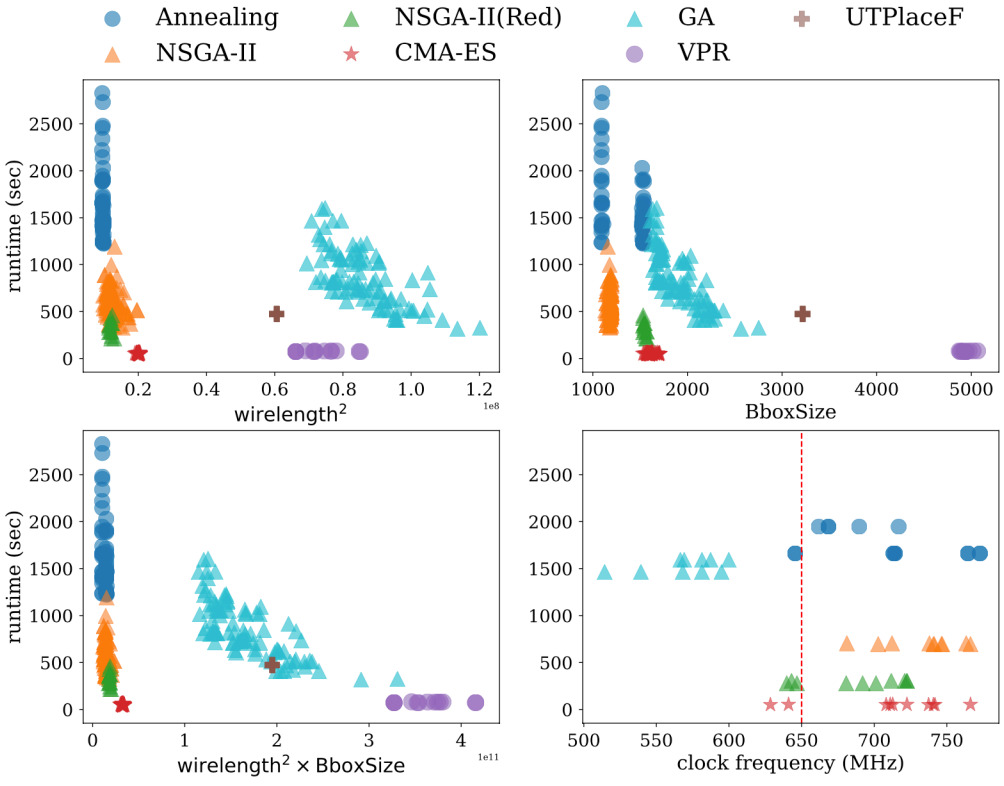
- RapidLayout accelerates the end-to-end workflow by 5–6 *times* against Vivado
- Eliminates the long-period manual effort of trail-and-error
- High quality placement results and pipelining achieves 650 MHz+ clock frequency

Experiment Results

QoR Comparison

- Optimized Simulated Annealing
- Verilog-to-Place-and-Route (VPR)
- UTPlaceF
- Single-objective Genetic Algorithm
- Manual Placement





Conclusions:

1. EA speeds up runtime over SA by 2.7–30.8×
2. EA wins wirelength over UTPlaceF and VPR by 1.8–2.4×, bbox size by 2.0–4.1×
3. EA achieves 750 MHz high freq. with 1–2 level of pipelining, NSGA-II saves ≈17K(6%) registers

Transfer Learning

Table 1: Transfer Learning Performance: VU3P, VU11P as seed devices

Device	Design Size (conv units)	Impl.Runtime (mins.)	Frequency (MHz)	Placement Runtime	
				Scratch (s)	Transfer (s)
xcvu3p	123	46.4	718.9	428.3	-
xcvu5p	246	56.9	677.9	396.0	55.7 (7.1×)
xcvu7p	246	55.1	670.2	345.4	44.2 (7.8×)
xcvu9p	369	58.4	684.9	316.5	45.5 (7×)
xcvu11p	480	65.2	655.3	695.9	-
xcvu13p	640	69.4	653.2	704.9	58.8 (12×)

- Devices with the same column number are applicable for transfer learning
- Transfer Learning accelerates optimization by 7–12×
- Frequency variation of transfer learning against from scratch: -2% – +7%

Conclusions

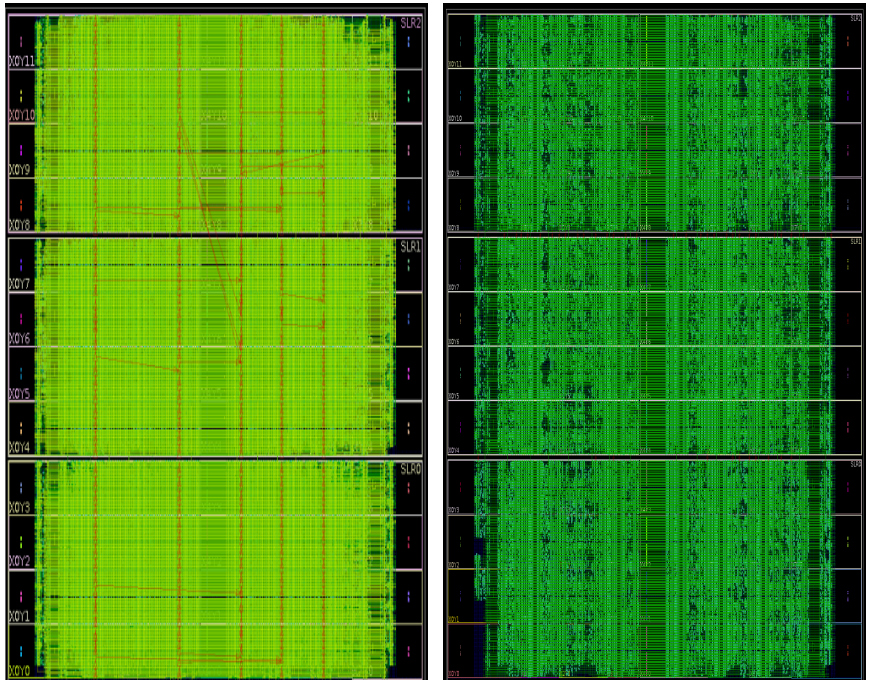
Conclusions

Key Contributions:

1. We solve hard-block-intense placement problem with evolutionary techniques
2. We build an end-to-end RapidLayout placement and routing toolflow
3. We develop the transfer learning process for hard block placement

Future Works:

1. Generalization: supporting any designs with hard blocks
2. GPU parallelization for evolutionary algorithms



(a) Vivado (b) RapidLayout

Figure 8: A Successful and Faster Placement Process by RapidLayout