UT DA

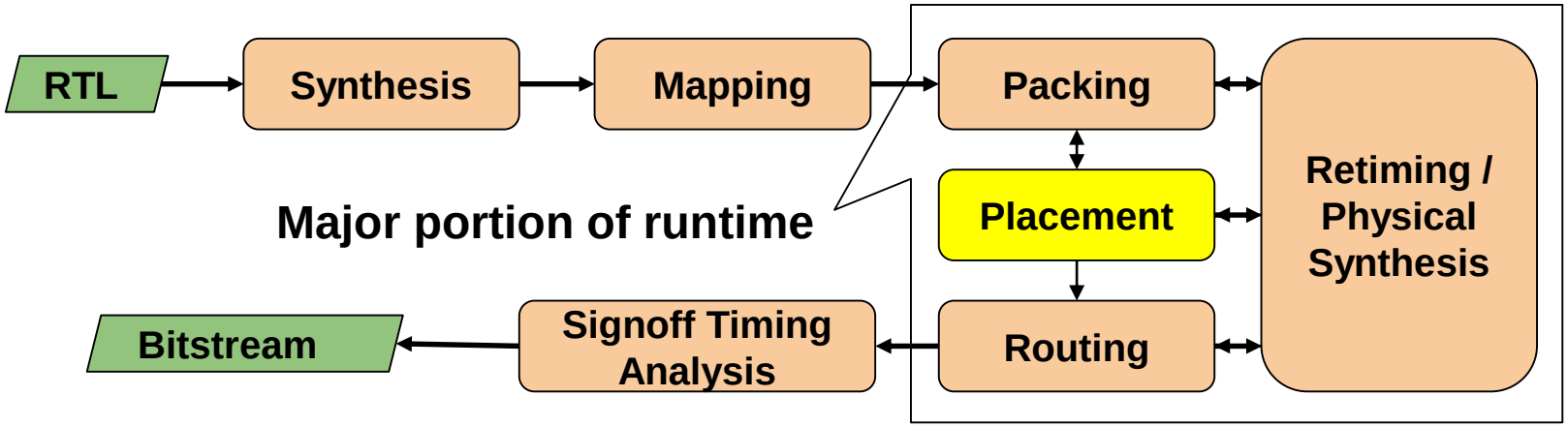
FPGA Accelerated FPGA Placement

Shounak Dhar[1], Love Singhal[2], Mahesh A. Iyer[2], David Z. Pan[1]

[1] The University of Texas at Austin
[2] Intel Corporation, San Jose

Work supported by Intel Strategic Research Alliance (ISRA)

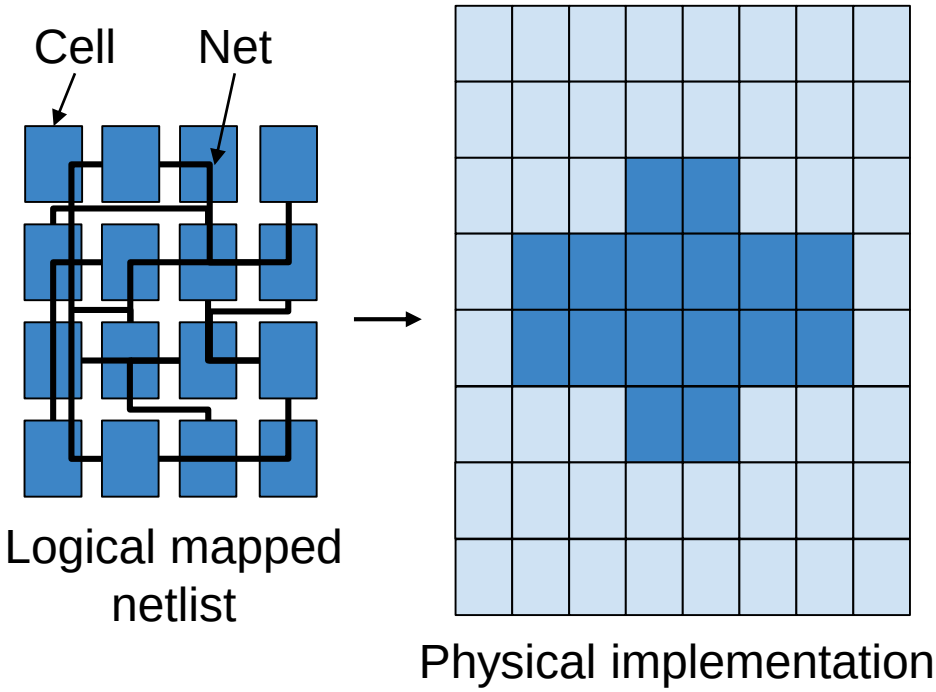
FPGA CAD Tool Flow

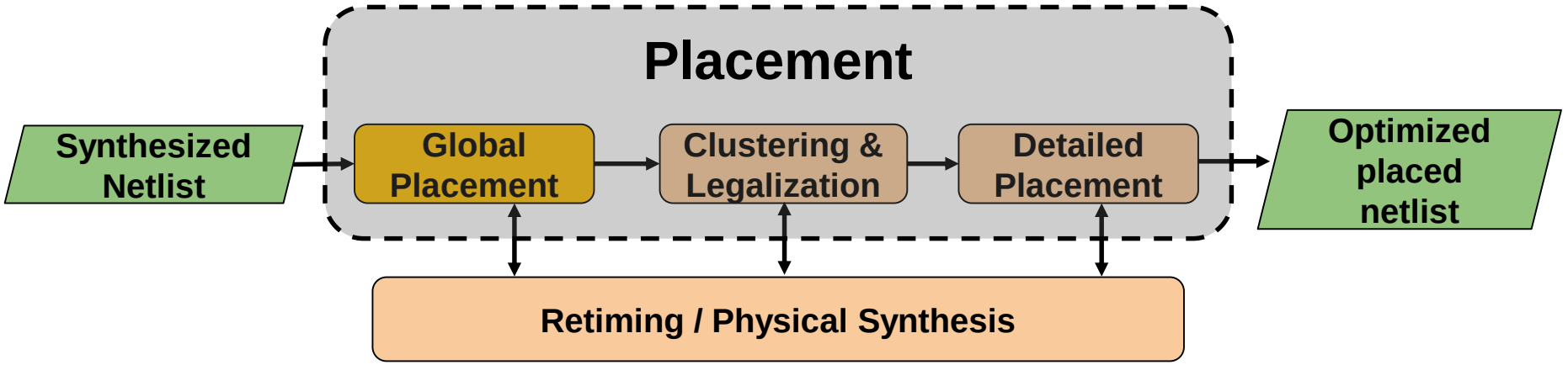


- Key metrics for FPGA competitive advantage:
  - Clock frequency (Fmax)
  - Power consumption
  - Compile time
    - Many NP-hard problems
  - Scaling with size and complexity of modern designs
- ***Focus: Accelerate FPGA placement***

FPGA Placement

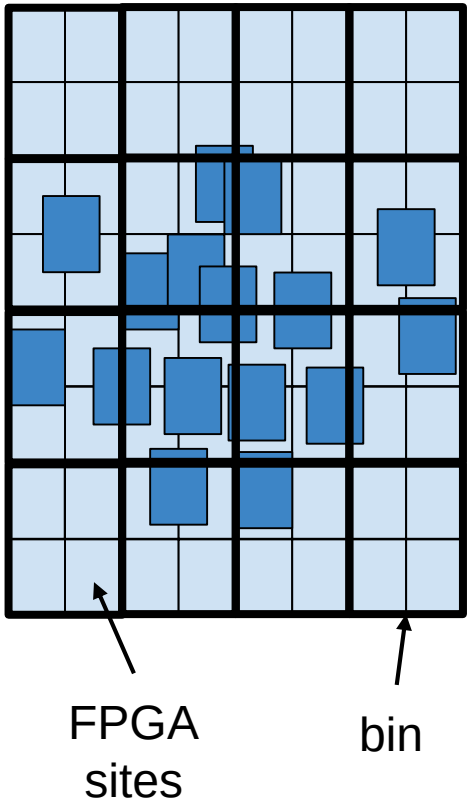
- Determines locations of components on a fixed-floorplan chip with limited resources (FPGA)
- Concurrent optimization of wirelength, timing, routing congestion, etc.





Global Placement

- Determines optimal (near-legal) locations of cells
- Analytical global optimization
  - Optimize wirelength, timing, congestion
  - $obj_1 = f(x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_n)$
  - Optimized using gradient descent
- Cell overlap minimization
  - Divide chip into small bins
  - $Overlap = demand - supply$



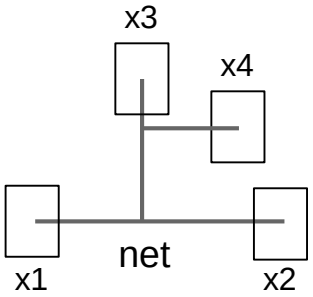
- Upper-Lower bound optimization
  - Iterate between global optimization and overlap minimization
  - Global objective with deviation penalty:

$$f(x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_n) + \lambda(\sum(x_i - x')^2 + \sum(y_i - y')^2)$$

Global Placement: Wirelength Model

- HPWL (Half-Perimeter WireLength) is a commonly used metric

$$xHPWL_{net} = \max_{u \in net}(x_u) - \min_{u \in net}(x_u)$$



xHPWL = x2-x1

- HPWL is not differentiable
  - Use a smooth approximation

$$WLx_{net} = \underbrace{\frac{\sum_{i \in net} x_i e^{\gamma x_i}}{\sum_{i \in net} e^{\gamma x_i}}}_{\text{soft max}} - \underbrace{\frac{\sum_{i \in net} x_i e^{-\gamma x_i}}{\sum_{i \in net} e^{-\gamma x_i}}}_{\text{soft min}}$$

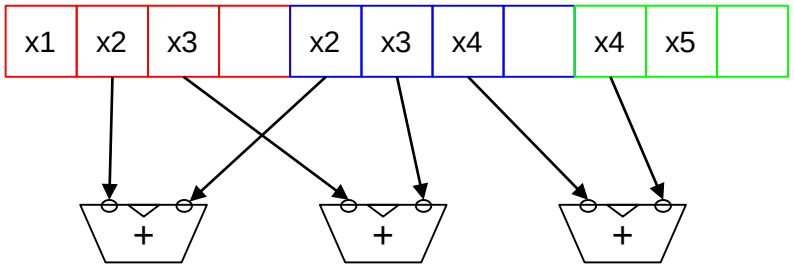
- γ controls smoothness of approximation
  - Higher $\gamma \rightarrow$ less smooth but more accurate
  - Lower $\gamma \rightarrow$ better for optimization convergence

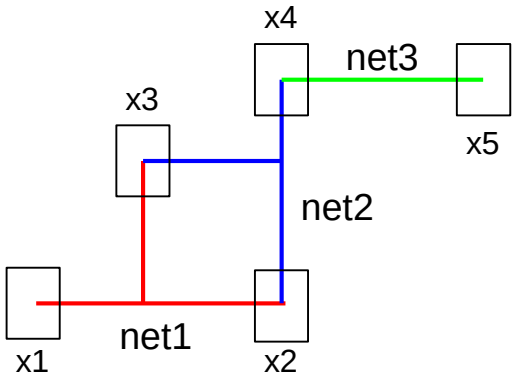
Global Placement: Runtime Bottleneck

- Wirelength gradient computation:
  - Step 1: compute gradient w.r.t each pin of each net

$$\frac{\partial(WLx_{net})}{\partial x_j} = \frac{(1+\gamma x_j)e^{\gamma x_j}}{\sum_{i\in net} e^{\gamma x_i}} - \frac{\gamma(\sum_{i\in net} x_i e^{\gamma x_i})e^{\gamma x_j}}{(\sum_{i\in net} e^{\gamma x_i})^2} - \frac{(1-\gamma x_j)e^{-\gamma x_j}}{\sum_{i\in net} e^{-\gamma x_i}} - \frac{\gamma(\sum_{i\in net} x_i e^{-\gamma x_i})e^{-\gamma x_j}}{(\sum_{i\in net} e^{-\gamma x_i})^2}$$

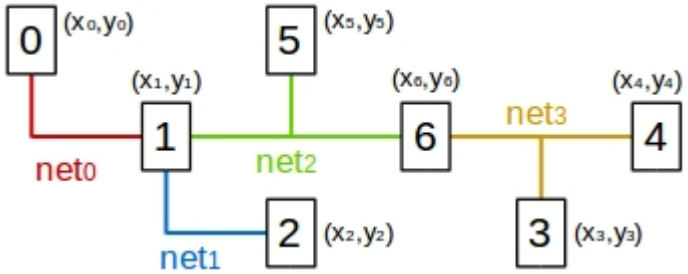
  - Simplify above equation by setting γ=1 and scaling x's
  - Step 2: add pin gradients to get cell gradients
    - Random memory accesses
    - Fitting all location data on FPGA is a challenge
    - Perform summation on CPU

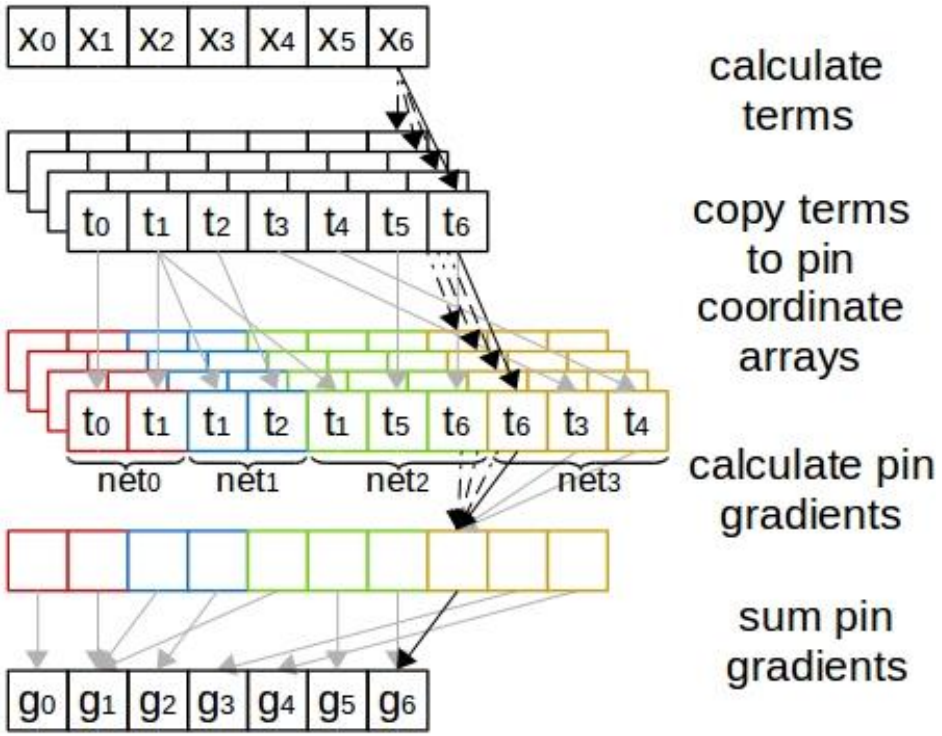




Baseline CPU Implementation

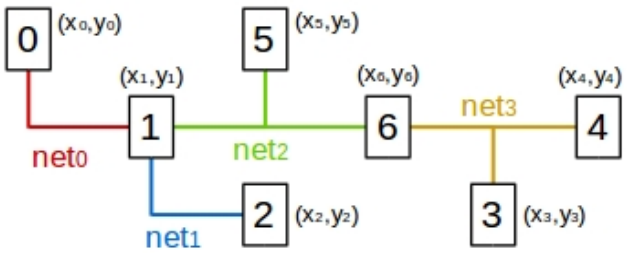
- Calculate 4 terms:
  - e^{x_i}, e^{-x_i}, $x_i e^{x_i}$, $x_i e^{x_i}$
- Calculate $\frac{d}{dx_i}$ for each pin
- Sum pin gradients to get cell gradients
- Multi-threaded and vectorized

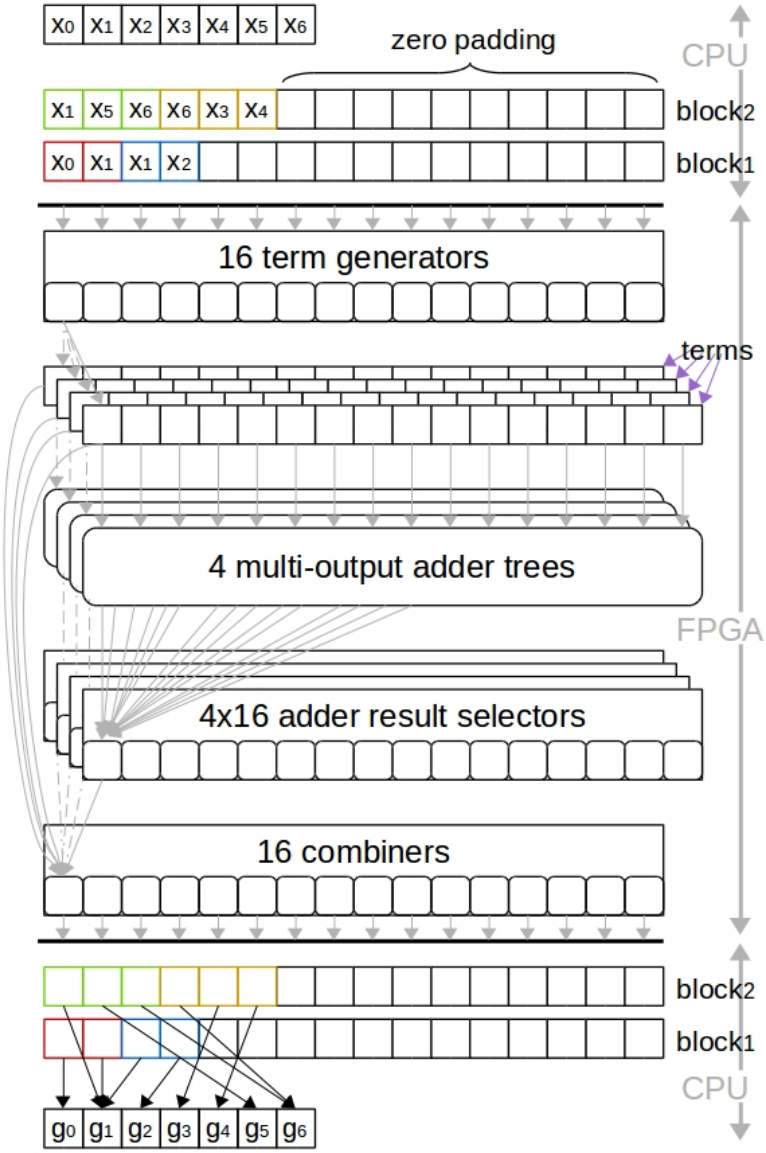




CPU+FPGA Implementation

- Sort nets by degree
  - Consider nets upto 16 pins
  - Group nets into blocks by degree
  - Each block has ≤ 16 pins
  - Padding to make each block 16 pins
- Compute pin gradients on FPGA
- Compute sum on CPU

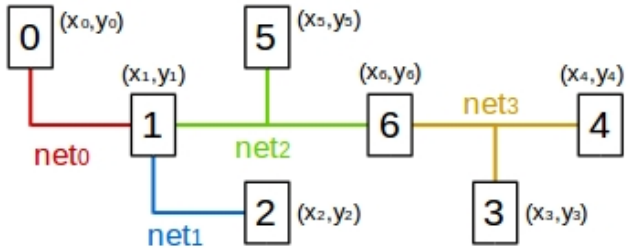


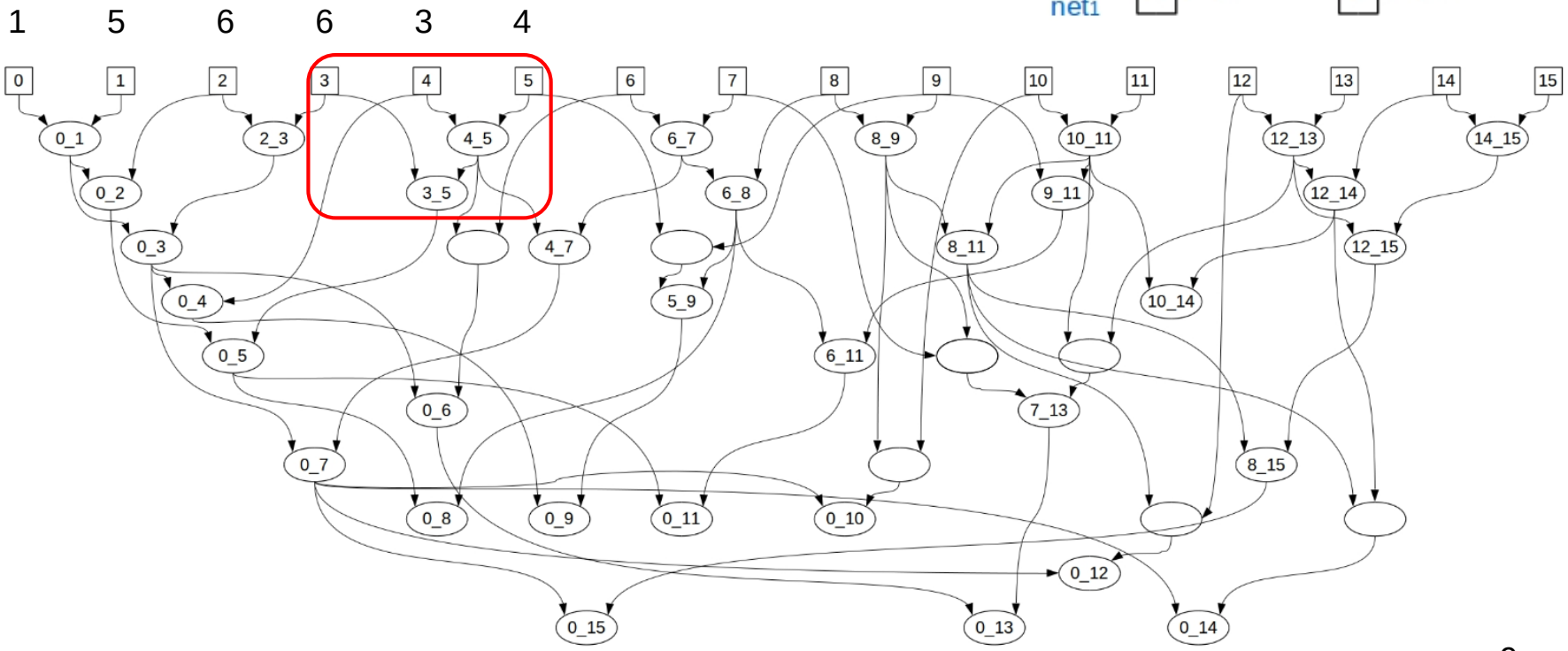


Adder Tree

- Multi-output adder tree to compute sums of terms
  - Sharing logic saves area

$$\frac{\partial(xWA_3)}{\partial x_6} = \frac{(1+x_6)e^{x_6}}{e^{x_6}+e^{x_3}+e^{x_4}} - \frac{(x_6e^{x_6}+x_3e^{x_3}+x_4e^{x_4})e^{x_6}}{(e^{x_6}+e^{x_3}+e^{x_4})^2}$$
$$-\frac{(1-x_6)e^{-x_6}}{e^{-x_6}+e^{-x_3}+e^{-x_4}} - \frac{(x_6e^{-x_6}+x_3e^{-x_3}+x_4e^{-x_4})e^{-x_6}}{(e^{-x_6}+e^{-x_3}+e^{-x_4})^2}$$





Benchmarks and Experimental Setup

Benchmarks

- ISPD 2016 FPGA placement contest

Design	# cells / 10^3	# nets / 10^3
FPGA01	105	105
FPGA02	166	167
FPGA03	421	428
FPGA04	423	430
FPGA05	425	433
FPGA06	704	713
FPGA07	707	716
FPGA08	717	725
FPGA09	867	876
FPGA10	952	961
FPGA11	845	851
FPGA12	1103	1111

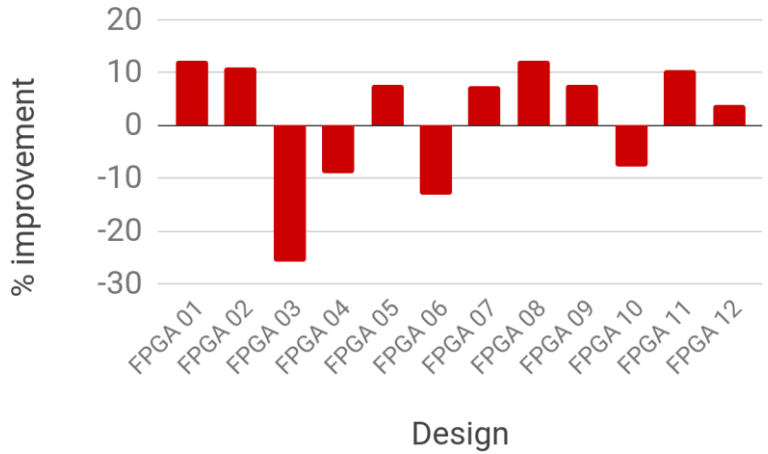
Experimental setup

- FPGA: Intel® Arria®10
- CPU: Intel® Xeon®; 14 cores, 28 threads
- Shared virtual memory; Low latency communication
- Compiled using Intel® FPGA SDK for OpenCL™ and Intel® Quartus® Prime Pro

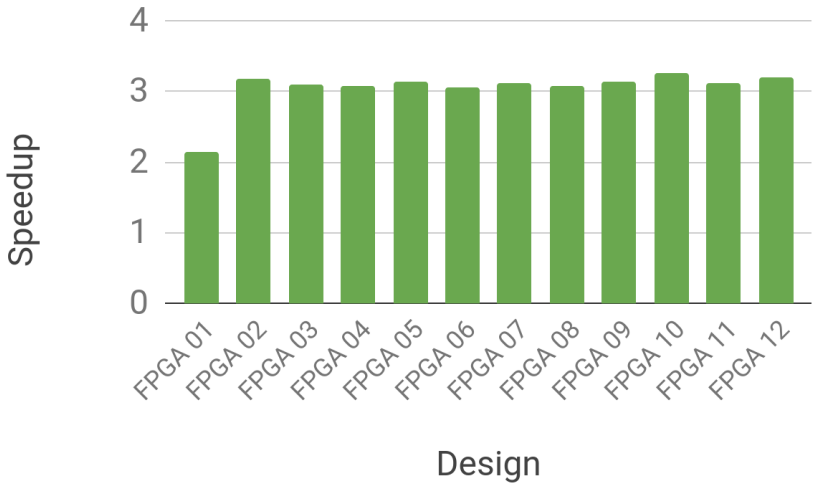
Logic	Register	RAM	DSP	Fmax
37%	32%	22%	67%	227 MHz

Results: Wirelength and Runtime

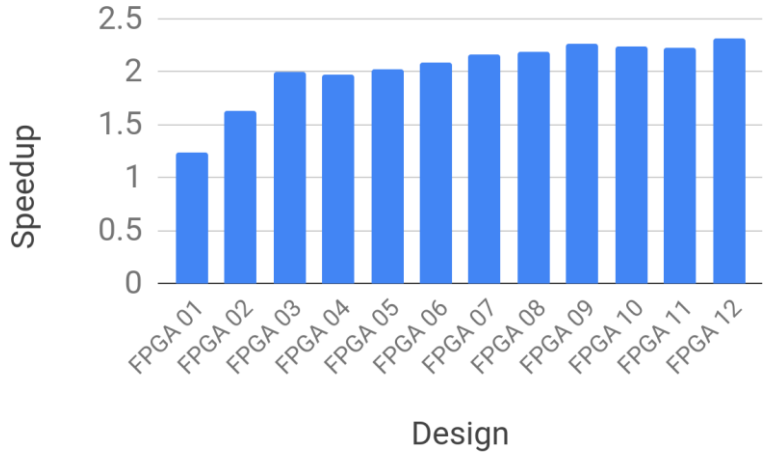
- 2.1% global placement wirelength improvement vs UTPlaceF



- 3.03x wirelength gradient speedup over CPU



- 2x global placement speedup over CPU



Thank You