Improving the Gradient Descent Based FPGA-Placement Algorithm

Tobias Thiemann Timm Bostelmann Sergei Sawitzki

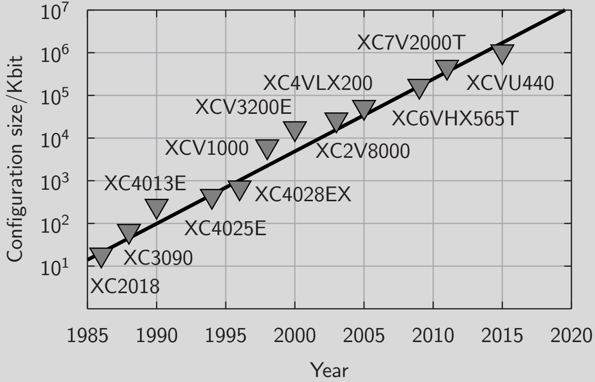
FH Wedel
University of Applied Sciences
Contact: bos@fh-wedel.de

IARIA
CENICS 2020

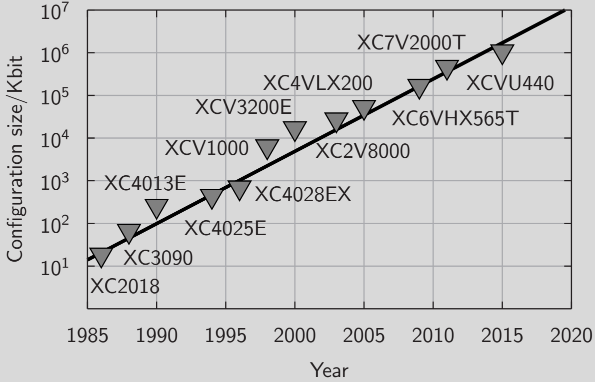
Presenter's Resume

Timm Bostelmann received his engineer's degree in computer engineering from the FH Wedel (University of Applied Sciences) in 2008. Since then, he is employed at FH Wedel as a research assistant in the field of embedded systems. In addition, he is working towards his PhD degree at the TU Dresden (University of Technology) in the field of reconfigurable architectures.

FPGA Complexity is Rising

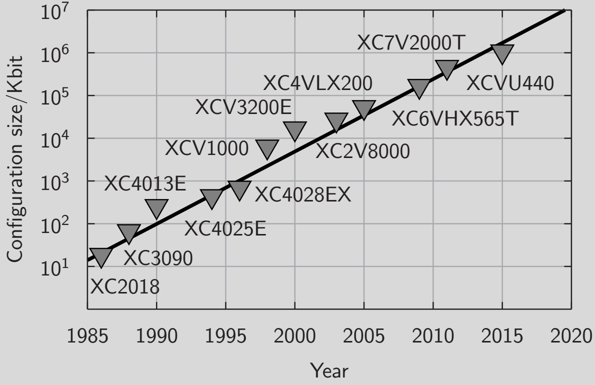


FPGA Complexity is Rising



EDA Tools have to handle this complexity

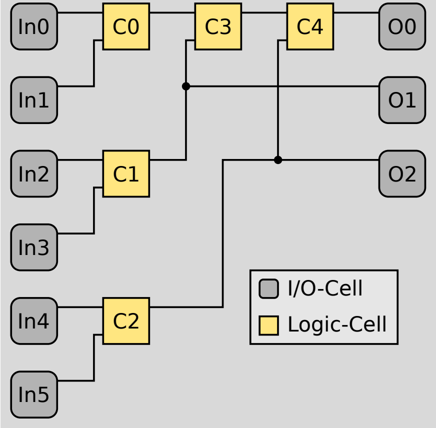
FPGA Complexity is Rising



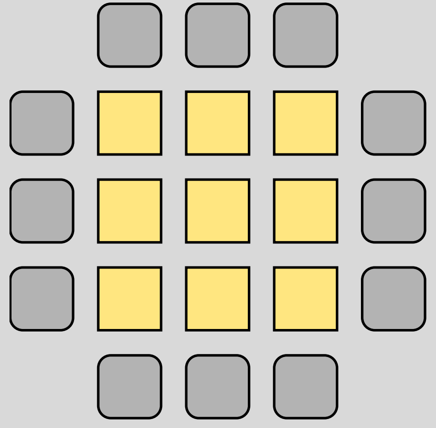
EDA Tools have to handle this complexity

Netlist Placement for FPGAs

Netlist

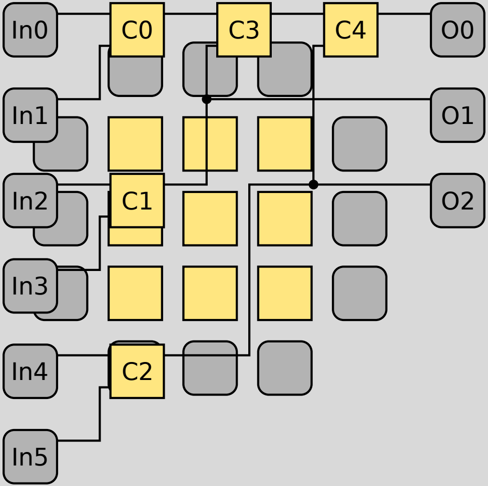


Architecture



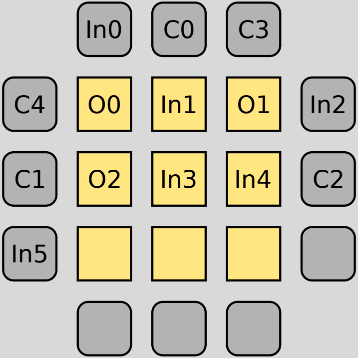
Netlist Placement for FPGAs

Placement — Illegal Positions



Netlist Placement for FPGAs

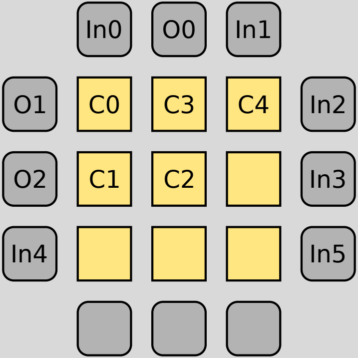
Placement — Illegal Types



- The cells are in the grid ...
- ... but the cell types are not compatible.

Netlist Placement for FPGAs

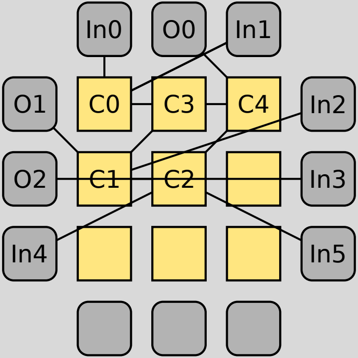
Placement — Legal



- The cells are in the grid ...
- ... and the cell types are compatible ...

Netlist Placement for FPGAs

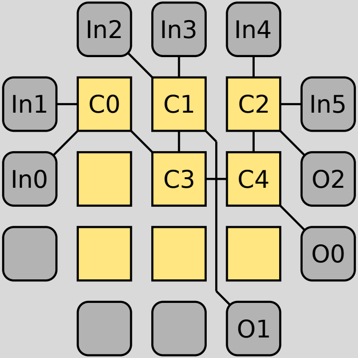
Placement — Legal



- The cells are in the grid ...
- ... and the cell types are compatible ...
- ... but the performance will be poor.

Netlist Placement for FPGAs

Placement — Good



- The cells are in the grid ...
- ...and the cell types are compatible ...
- ...and the performance will be good.

Netlist Placement for FPGAs

Problem Description

Select a resource cell on the target FPGA for every cell of the given netlist in a way that:

1. Every cell of the netlist is assigned to a resource cell of the fitting type (e.g IO, CLB, DSP)
2. No resource cell is occupied by more than one cell of the netlist
3. The cells are arranged in a way that allows the best possible routing

Netlist Placement for FPGAs

Problem Description

Select a resource cell on the target FPGA for every cell of the given netlist in a way that:

1. Every cell of the netlist is assigned to a resource cell of the fitting type (e.g IO, CLB, DSP)
2. No resource cell is occupied by more than one cell of the netlist
3. The cells are arranged in a way that allows the best possible routing

Established Solutions

- Iterative algorithms like Simulated Annealing
- Constructive algorithms like min-cut (recursive partitioning)
- Analytical placement

Previous Work

Fast FPGA-Placement Using a Gradient Descent Based Algorithm

- Achieved similar results to the reference (based on simulated annealing) regarding the bounding-box quality
- Is on average 3.8 times faster then the reference
- Results in a significantly longer critical path
- Is working single threaded

Previous Work

Fast FPGA-Placement Using a Gradient Descent Based Algorithm

- Achieved similar results to the reference (based on simulated annealing) regarding the bounding-box quality
- Is on average 3.8 times faster then the reference
- Results in a significantly longer critical path
- Is working single threaded

This Work

- Different approaches to reduce the length of the critical path are evaluated
- Different approaches to reduce the runtime (including parallelization) are evaluated
- Extensive benchmarking

General Approach

- Measure the quality of the placement with a cost function
- Move all nodes towards the steepest gradient descent
- Legalize the placement
- Repeat optimization and legalization in a loop

General Approach

- Measure the quality of the placement with a cost function
- Move all nodes towards the steepest gradient descent
- Legalize the placement
- Repeat optimization and legalization in a loop

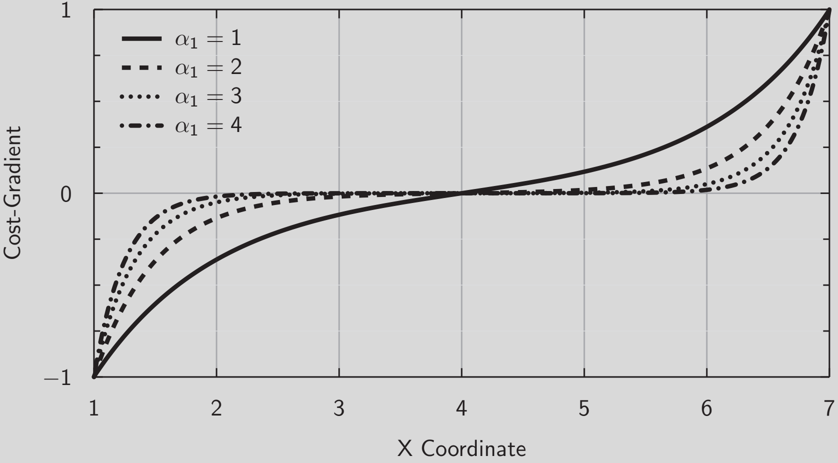
Cost Function

An exponential function over the distance between the position of the node and the bounding-box of the net is chosen as basis of the cost-function:

$$C_k = \alpha_2 \cdot \sum_{n \in N_k} \Big(e^{\alpha_1 \cdot (x_k - \max_x(n))} + e^{\alpha_1 \cdot (\min_x(n) - x_k)} + e^{\alpha_1 \cdot (y_k - \max_y(n))} + e^{\alpha_1 \cdot (\min_y(n) - y_k)} \Big)$$

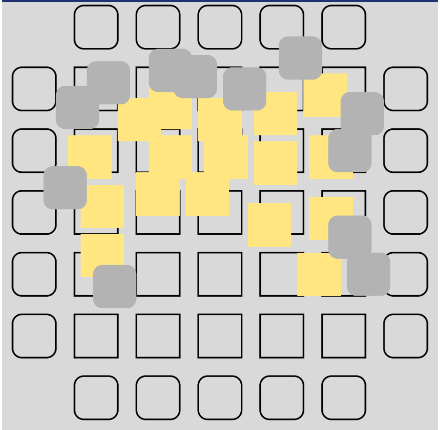
Cost-Gradient

Plot of the gradient for the X coordinate of a node, assuming a net with the boundaries $min_x = 1$ and $max_x = 7$:

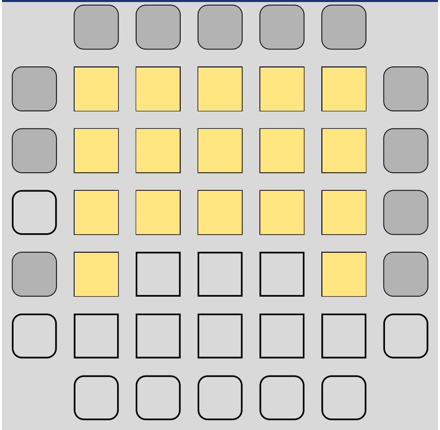


Legalization

Illegal Placement



Legal Placement



Placement Phases

1. Presorting (5000 iterations)
   high step width, weak legalization
2. Grid placement (1000 iterations)
   high step width, stronger legalization
3. Initial detailed placement (1000 iterations)
   reduced step width
4. Detailed placement (5000 iterations)
   reduced optimization step width
5. Final placement (100 iterations)
   no optimization, only legalization

Evaluated Approaches

1. Utilization of multithreading
   The algorithm was profiled and a parallelized implementation was derived
2. Improvement of the initial placement
   The initial placement was generated with a min-cut approach instead of a random initialization
3. Improvement of the critical path
   A path metric was introduced to favor nodes on long paths
4. Optimization of the parameters
   The parameters of the algorithm were optimized using an artificial neural network

Benchmarking Setup

- The original gradient algorithm (GPO), the new gradient algorithm (GPN) and simulated annealing (VPR) are compared
- All measurements are done for twenty common netlists
- Non deterministic values are averaged over ten measurements

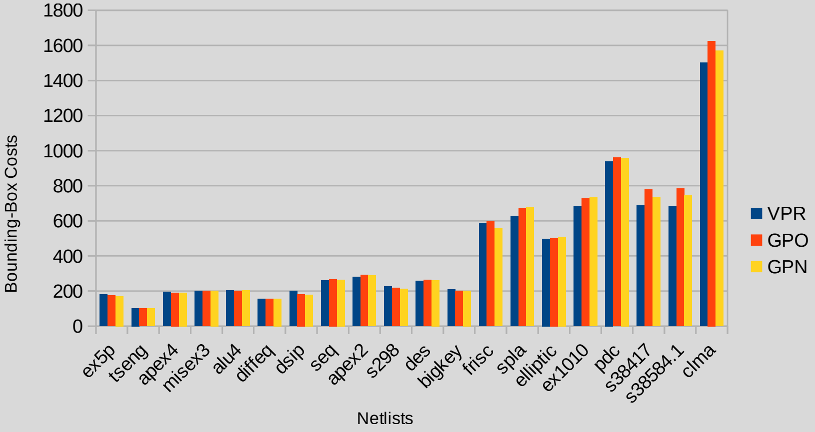
Benchmarking Setup

- The original gradient algorithm (GPO), the new gradient algorithm (GPN) and simulated annealing (VPR) are compared
- All measurements are done for twenty common netlists
- Non deterministic values are averaged over ten measurements

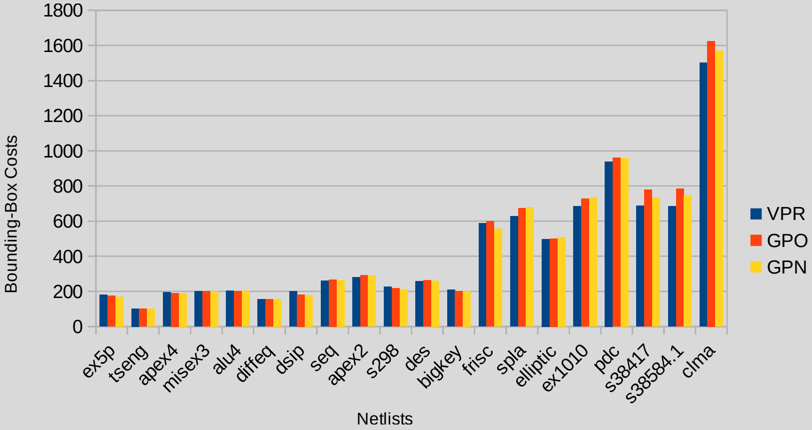
Measurement Series

1. Bounding-Box Costs
2. Critical Path
3. Runtime

Comparison of the Bounding-Box Costs



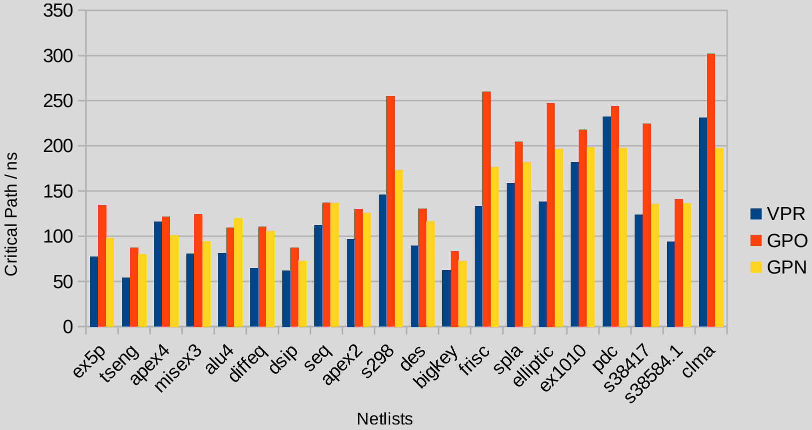
Comparison of the Bounding-Box Costs



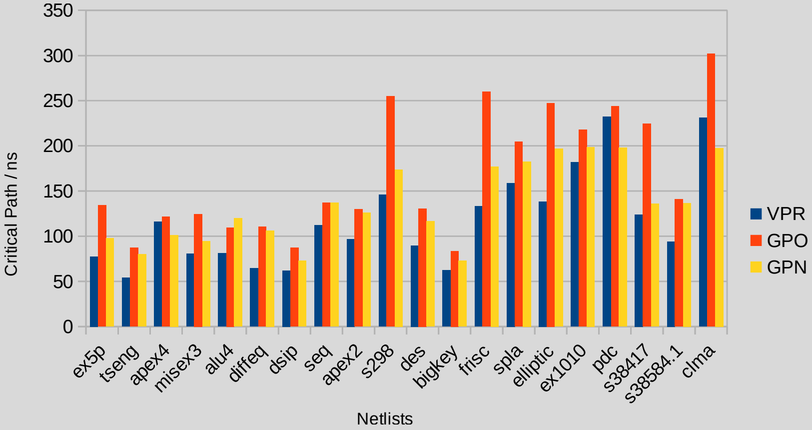
$$\text{Average}\left(\frac{\text{bbcost}(GPN)}{\text{bbcost}(VPR)} \times 100\,\%\right) = 100.57\,\%$$

$$\text{Average}\left(\frac{\text{bbcost}(GPN)}{\text{bbcost}(GPO)} \times 100\,\%\right) = 98.73\,\%$$

Comparison of the Critical Path



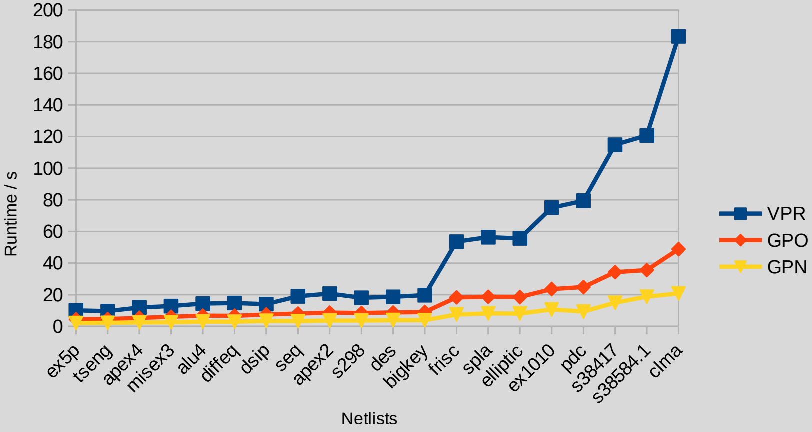
Comparison of the Critical Path



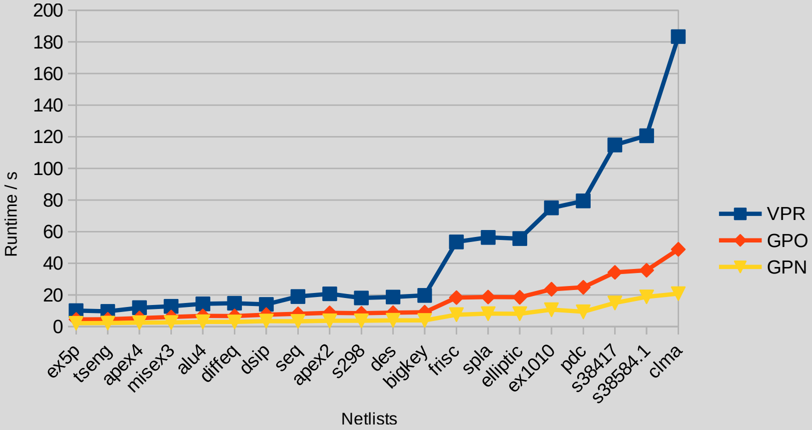
$$\text{Average}\left(\frac{\text{critpath}(GPN)}{\text{critpath}(VPR)} \times 100\,\%\right) = 121.14\,\%$$

$$\text{Average}\left(\frac{\text{critpath}(GPN)}{\text{critpath}(GPO)} \times 100\,\%\right) = 84.00\,\%$$

Comparison of the Runtime



Comparison of the Runtime



$$\text{Average}\left(\frac{\text{runtime}(GPN)}{\text{runtime}(VPR)} \times 100\,\%\right) = 19.69\,\%$$

$$\text{Average}\left(\frac{\text{runtime}(GPN)}{\text{runtime}(GPO)} \times 100\,\%\right) = 46.39\,\%$$

Conclusion

- The new gradient algorithm is about 5 times as fast as VPR and more than two times as fast as the original gradient algorithm
- The bounding box quality is about equal for all three algorithms
- That critical path of the new gradient algorithm is about 20 % longer compared to VPR and about 16 % shorter compared to the original gradient algorithm
- Extended benchmarking with even larger netlists might underline the scalability of the approach

Conclusion

- The new gradient algorithm is about 5 times as fast as VPR and more than two times as fast as the original gradient algorithm
- The bounding box quality is about equal for all three algorithms
- That critical path of the new gradient algorithm is about 20 % longer compared to VPR and about 16 % shorter compared to the original gradient algorithm
- Extended benchmarking with even larger netlists might underline the scalability of the approach

Thank you for your attention!