A Deep Learning Framework to Predict Routability for FPGA Circuit Placement

A. Al-Hyari, A. Shamli, Z. Abuowaimer, G. Grewal, and S. Areibi
Guelph FPGA CAD Group
https://fpga.socs.uoguelph.ca
Motivation

Two major problems:

- **Place-and-Route** are the two most time-consuming steps in the FPGA CAD flow.
- **Placement** solutions produced by a placement tool may be unroutable.

Fast and Accurate Routability Prediction:

- Ability for placer to respond early and often to improve P&R runtimes and increase likelihood of producing a **routable** placement.
Outline

- Background
- Deep Learning Framework
- Results
- Case Study
- Conclusions & Future work
Placement Problem

• Given a circuit in the form of a netlist, map the components in the netlist onto locations (resources) on the FPGA such that:
  • Minimize objectives ➞ wirelength, delay, congestion, etc
  • Subject to several constraints: based on architecture of FPGA

• Target: Xilinx Ultrascale
Modern FPGA Architecture

- Architecture of modern FPGA devices imposes additional constraints on placement problem.
- Modern FPGAs are heterogeneous.
- Slice architecture imposes constraints on packing and placement.
### Benchmarks

<table>
<thead>
<tr>
<th>Design</th>
<th>#LUTs (util)</th>
<th>#Flops (util)</th>
<th>#BRAMs</th>
<th>#DSPs</th>
<th>#control sets</th>
<th>Rent</th>
<th>Exponent</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPGA-1</td>
<td>50K (9%)</td>
<td>55K (5%)</td>
<td>0 (0%)</td>
<td>0 (0%)</td>
<td>12</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>FPGA-2</td>
<td>100K (19%)</td>
<td>66K (6%)</td>
<td>100 (6%)</td>
<td>100 (13%)</td>
<td>121</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>FPGA-3</td>
<td>250K (47%)</td>
<td>170K (16%)</td>
<td>600 (35%)</td>
<td>500 (65%)</td>
<td>1281</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>FPGA-4</td>
<td>250K (47%)</td>
<td>172K (16%)</td>
<td>600 (35%)</td>
<td>500 (65%)</td>
<td>1281</td>
<td>0.7</td>
<td></td>
</tr>
<tr>
<td>FPGA-5</td>
<td>250K (47%)</td>
<td>174K (16%)</td>
<td>600 (35%)</td>
<td>500 (65%)</td>
<td>1281</td>
<td>0.8</td>
<td></td>
</tr>
<tr>
<td>FPGA-6</td>
<td>350K (65%)</td>
<td>352K (33%)</td>
<td>1000 (58%)</td>
<td>600 (78%)</td>
<td>2541</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>FPGA-7</td>
<td>350K (65%)</td>
<td>355K (33%)</td>
<td>1000 (58%)</td>
<td>600 (78%)</td>
<td>2541</td>
<td>0.7</td>
<td></td>
</tr>
<tr>
<td>FPGA-8</td>
<td>500K (93%)</td>
<td>216K (20%)</td>
<td>600 (35%)</td>
<td>500 (65%)</td>
<td>1281</td>
<td>0.7</td>
<td></td>
</tr>
<tr>
<td>FPGA-9</td>
<td>500K (93%)</td>
<td>366K (34%)</td>
<td>1000 (58%)</td>
<td>600 (78%)</td>
<td>2541</td>
<td>0.7</td>
<td></td>
</tr>
<tr>
<td>FPGA-10</td>
<td>350K (65%)</td>
<td>600K (56%)</td>
<td>1000 (58%)</td>
<td>600 (78%)</td>
<td>2541</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>FPGA-11</td>
<td>480K (89%)</td>
<td>363K (34%)</td>
<td>1000 (58%)</td>
<td>400 (52%)</td>
<td>2091</td>
<td>0.7</td>
<td></td>
</tr>
<tr>
<td>FPGA-12</td>
<td>500K (93%)</td>
<td>602K (56%)</td>
<td>600 (35%)</td>
<td>500 (65%)</td>
<td>1281</td>
<td>0.6</td>
<td></td>
</tr>
</tbody>
</table>

- **The 12 ISPD 2016 routing-aware placement contest circuits**

- **Our industrial partner, Xilinx Inc., synthesized an extra 360 benchmarks using an internal netlist generation tool**

<table>
<thead>
<tr>
<th>#LUTs</th>
<th>#FFs</th>
<th>#BRAMs</th>
<th>#DSPs</th>
<th>#CSETs</th>
<th>#IOs</th>
<th>Rent Exp</th>
</tr>
</thead>
<tbody>
<tr>
<td>44K – 518K</td>
<td>52K – 630K</td>
<td>0 - 1035</td>
<td>0 - 620</td>
<td>11 - 2684</td>
<td>150 - 600</td>
<td>0.4 – 0.8</td>
</tr>
</tbody>
</table>
GPlace3.0 Flow

**PHASE I**
1. Pin Propagation Pre-placement
2. WL-driven Global Placement
   - Star+ Jacobi Solver
   - Window-Based Legalization
     - LUT Sharing
     - LUT Bipartitioning
     - FF Bipartitioning
     - DSP Bipartitioning
     - BRAM Bipartitioning

**PHASE II**
3. Congestion Estimation
4. Cell (LUT) Inflation
5. Congestion-driven Global Placement
   - Star+ Jacobi Solver
   - Window-Based Legalization
     - LUT Sharing
     - LUT Bipartitioning (with LUT Density)
     - FF Bipartitioning
     - DSP Bipartitioning
     - BRAM Bipartitioning

**PHASE III**
6. Detailed Placement (DOISM)
   - ISM (minimize HPWL)
   - ISM (min. external pins)

---

Flat Netlist → Final Placement
GPlace3.0 minimizes both wirelength and congestion over time, but without any knowledge of whether or not the solution is routable or not.
Benefits of Predicting Routability

- Accurate Routability Prediction
  - Increase number of iterations, if probability to route is low
  - If not routable, perform further optimization
  - Proceed with routing stage

GPlace
- WL-driven global placement
- Congestion-driven global placement
- Detailed placement
The ability to accurately and efficiently estimate the routability of a circuit based on its placement is one of the most challenging tasks in the FPGA flow.

Providing an informative feedback about the routability can help the placement tool to further enhance its optimization strategy.

Questions:

- Is it possible to develop a framework that is placer independent and architecture independent?
- Can Deep Learning be used to predict the routability of a placement?
Outline for Reminder of Talk

- Background
- Deep Learning Framework
- Results
- Case Study
- Conclusion
Machine learning is a type of Artificial Intelligence (AI) that can provide systems with the ability to learn without being explicitly programmed.
Supervised Machine Learning

\[ y = f(x) \]

- **Training**: given a \textit{training set} of labeled examples \{\((x_1, y_1), \ldots, (x_N, y_N)\}\}, estimate the prediction function \(f()\) by minimizing the prediction error on the training set.

- **Testing**: apply \(f()\) to a never before seen \textit{test example} \(x\) and output the \textit{predicted value} \(y = f(x)\)
• Deep learning (DL) is a subset of machine learning, **it refers to having deeper hierarchy in a neural network.**

• DL is capable of processing high dimensional data.

**Shallow network**

**Deep network**
Convolutional Neural Network (CNN)

- CNN is the most commonly used form of DL because it can process an image and generate a meaningful response depending on the application.
- Convolutional filters are capable of capturing the spatial relationship between surrounding elements implicitly.
Features

- Four features are calculated for each G-Cell (corresponds to a switch box) of the FPGA.
- Each feature is designed to characterize the routing resource utilization of the switch.

\[
\begin{align*}
  f_1 &= \text{WLPA} \\
  f_2 &= \text{Pin density} \\
  f_3 &= \text{NCPR}_{5x5} \\
  f_4 &= \text{NCPR}_{9x9}
\end{align*}
\]
Features

- Four features are calculated for each G-Cell (corresponds to a switch box) of the FPGA
- Each feature is designed to characterize the routing resource utilization of the switch

\[ f_1 = \sum_{n \in N_t} \frac{w_n \cdot HPWL_n}{\#gcell_n} \]

\[ w_n = 1 \]
\[ HPWL_n = 5 \]
\[ \#gcell_n = 12 \]
Features

- Four features are calculated for each G-Cell (corresponds to a switch box) of the FPGA
- Each feature is designed to characterize the routing resource utilization of the switch

\[
f_1 = \sum_{n \in N_t} \frac{w_n \cdot HPWL_n}{\#_gcell_n}
\]

\[
f_2 = \sum_{n \in N_t} \#_pins_{n, gcell_i}
\]

\[
f_3 = |W_{5x5}|
\]

\[
f_4 = |W_{9x9}|
\]

\[\#_{pins_{n, gcell_i}} = 3\]

\[|W_{3x3}| = 2\]
DLRoute: Overall Methodology

1. Placement
2. Congestion Features (Local): WLPA, Pin Count, NCPR$_{5\times5}$, NCPR$_{9\times9}$
3. MLCong Congestion Estimate [5]
4. Congestion Map
5. DLRout Route Routability Prediction
6. Routability Label {0,1}
Heat-maps change across the three phases of placement flow.

- **PHASE1**: Global (Wirelength) Placement
- **PHASE2**: Global (Congestion) Placement
- **PHASE3**: Detailed Placement

*Iterations*
DLRoute Framework for Routability Prediction

- Gplace3.0 is used to place all the 372 Xilinx benchmarks, a placement file is saved after every iteration throughout the placement.
- A total of 26551 placements are generated and saved.
- The four features of congestion (same as MLCong) are extracted.
- The four feature of each placement are used to generate a heatmap.
- A total 26551 heatmaps are collected.
- Vivado router is used to route all the generated placement files.
- The labels of the heatmaps are determined by Vivado router, each heatmap has a binary label {0, 1} that reflects the routability of this heatmap.
- The heatmaps are splitted into two sets, a training set that represents (70%) of the total heatmaps, while the testing set represents (30%) of the total heatmaps.
- The training data is used to train the CNN model offline, while the testing set is used to assess the accuracy of the model.
- The CNN model with the best accuracy is saved to be used later in deployment stage.
- In deployment phase, the four features of congestion are extracted from a new placement file. Then, a heatmap is generated from these features.
- The saved CNN model with best performance is used to generate the routability label for this new placement.
A heat-map of size 480x168 represents the congestion in a placement as the input to the CNN.

- A convolutional layer (CONV1) of 32 filters with size of (6,3) and stride of (2,1) with max-pooling is applied to the input heat map. The activation function is ReLU. The generated feature maps are 238x166 in dimension.
- A convolutional layer (CONV2) of 32 filters with size of (3,3) and stride of (2,2) with max-pooling is applied to the output of CONV1 layer. The activation function is ReLU. The generated feature maps are 79x83 in dimension.
- A convolutional layer (CONV3) of 32 filters with size of (3,3) and stride of (1,1) with max-pooling is applied to the output of CONV2 layer. The activation function is ReLU. The generated feature maps are 19x20 in dimension.
- A convolutional layer (CONV4) of 32 filters with size of (3,3) and stride of (1,1) with max-pooling is applied to the output of CONV3 layer. The activation function is ReLU. The generated feature maps are 4x4 in dimension.

A flatten layer converts the two-dimensional feature map into a one-dimensional vector. Two fully-connected layers are applied to classify the vector of the flatten layer. Each layer has 100 neurons. The activation function is ReLU. A binary label \( \{0, 1\} \) is generated by the output neuron in the output layer. The activation function is sigmoid.
- The network takes a congestion heat map of size 480x168
- Four convolutional layers with a depth of 32 filters are used to extract features
- Two fully connected layers are used to classify the flattened vector of features
- A sigmoid output neuron generates a binary label of \{0, 1\} as routability label
## DLRoute: Architecture Details

### Optimized CNN Architecture

<table>
<thead>
<tr>
<th>#</th>
<th>Architecture Parameters</th>
<th>Output Volume</th>
<th># of Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CONV1: 32x(6,3), stride: (2,1)</td>
<td>238x166x32</td>
<td>1760</td>
</tr>
<tr>
<td>2</td>
<td>Maxpool: (3,2), stride: (3,2)</td>
<td>79x83x32</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>CONV2: 32x(3,3), stride: (2,2)</td>
<td>39x41x32</td>
<td>9248</td>
</tr>
<tr>
<td>4</td>
<td>Maxpool: (2,2), stride: (2,2)</td>
<td>19x20x32</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>CONV3: 32x(3,3), stride: (2,2)</td>
<td>9x9x32</td>
<td>9248</td>
</tr>
<tr>
<td>6</td>
<td>Maxpool: (2,2), stride: (2,2)</td>
<td>4x4x32</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>CONV4: 32x(3,3), stride: (1,1)</td>
<td>2x2x32</td>
<td>9248</td>
</tr>
<tr>
<td>8</td>
<td>Flatten</td>
<td>128</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>FC1: 100 (ReLU)</td>
<td>100</td>
<td>12900</td>
</tr>
<tr>
<td>10</td>
<td>FC2: 100 (ReLU)</td>
<td>100</td>
<td>10100</td>
</tr>
<tr>
<td>11</td>
<td>OUT: 1 (Sigmoid)</td>
<td>1</td>
<td>101</td>
</tr>
</tbody>
</table>

**Total number of parameters**: 52605

### Best Hyper-Parameters

<table>
<thead>
<tr>
<th>Hyper-Parameters</th>
<th>Training Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Optimizer</strong></td>
<td><strong>Learning Rate</strong></td>
</tr>
<tr>
<td>ADAM</td>
<td>0.0001</td>
</tr>
</tbody>
</table>
The curves clearly show that the model is neither under-fitting nor over-fitting the data.
# DLRroute: Performance Results

### Overall Performance

<table>
<thead>
<tr>
<th>Accuracy</th>
<th>Precision</th>
<th>Sensitivity</th>
<th>Specificity</th>
<th>M</th>
<th>Train Time</th>
<th>Test Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>97.4%</td>
<td>0.961</td>
<td>0.980</td>
<td>0.970</td>
<td>0.876</td>
<td>115.8 (min)</td>
<td>7.8 (ms)</td>
</tr>
</tbody>
</table>

### Performance on Each Placement Phase

<table>
<thead>
<tr>
<th>Phase</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Sensitivity</th>
<th>Specificity</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global (Wirelength) Placement</td>
<td>0.988</td>
<td>0.955</td>
<td>0.993</td>
<td>0.986</td>
<td>0.967</td>
</tr>
<tr>
<td>Global (Congestion) Placement</td>
<td>0.958</td>
<td>0.944</td>
<td>0.962</td>
<td>0.954</td>
<td>0.915</td>
</tr>
<tr>
<td>Detailed Placement</td>
<td>0.983</td>
<td>0.987</td>
<td>0.995</td>
<td>0.826</td>
<td>0.864</td>
</tr>
</tbody>
</table>
### Performance on Each Benchmark

<table>
<thead>
<tr>
<th>Set</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Sensitivity</th>
<th>Specificity</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPGA1</td>
<td>0.962</td>
<td>0.968</td>
<td>0.986</td>
<td>0.864</td>
<td>0.876</td>
</tr>
<tr>
<td>FPGA2</td>
<td>0.975</td>
<td>0.980</td>
<td>0.990</td>
<td>0.907</td>
<td>0.913</td>
</tr>
<tr>
<td>FPGA3</td>
<td>0.988</td>
<td>0.987</td>
<td>0.996</td>
<td>0.970</td>
<td>0.971</td>
</tr>
<tr>
<td>FPGA4</td>
<td>0.976</td>
<td>0.964</td>
<td>0.988</td>
<td>0.966</td>
<td>0.953</td>
</tr>
<tr>
<td>FPGA5</td>
<td>0.990</td>
<td>0.947</td>
<td>0.991</td>
<td>0.990</td>
<td>0.963</td>
</tr>
<tr>
<td>FPGA6</td>
<td>0.987</td>
<td>0.974</td>
<td>0.992</td>
<td>0.983</td>
<td>0.972</td>
</tr>
<tr>
<td>FPGA7</td>
<td>0.899</td>
<td>0.813</td>
<td>0.888</td>
<td>0.903</td>
<td>0.774</td>
</tr>
<tr>
<td>FPGA8</td>
<td>0.987</td>
<td>0.985</td>
<td>0.992</td>
<td>0.979</td>
<td>0.973</td>
</tr>
<tr>
<td>FPGA9</td>
<td>0.991</td>
<td>0.965</td>
<td>0.993</td>
<td>0.991</td>
<td>0.973</td>
</tr>
<tr>
<td>FPGA10</td>
<td>0.969</td>
<td>0.913</td>
<td>0.990</td>
<td>0.960</td>
<td>0.929</td>
</tr>
<tr>
<td>FPGA11</td>
<td>0.988</td>
<td>0.981</td>
<td>0.981</td>
<td>0.991</td>
<td>0.972</td>
</tr>
<tr>
<td>FPGA12</td>
<td>0.979</td>
<td>0.992</td>
<td>0.909</td>
<td>0.998</td>
<td>0.937</td>
</tr>
</tbody>
</table>
Outline for Reminder of Talk

Outline

- Background
- Deep Learning Framework
- Results
- Case Study
- Conclusion
The proposed routability predictor can be used to avoid costly, and futile place-and-route iterations.

The savings in time ranges from 42.7% to 82.1%.

<table>
<thead>
<tr>
<th>Placer</th>
<th>Routability</th>
<th>CPU Time</th>
<th>Saving</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Routable</td>
<td>Non-Routable</td>
<td>Routed</td>
</tr>
<tr>
<td>UTPlace[8]</td>
<td>317 (85%)</td>
<td>55 (15%)</td>
<td>315473</td>
</tr>
<tr>
<td>Ripple[9]</td>
<td>336 (90%)</td>
<td>36 (10%)</td>
<td>338626</td>
</tr>
<tr>
<td>Vivado2015.4</td>
<td>262 (70%)</td>
<td>110 (30%)</td>
<td>209402</td>
</tr>
<tr>
<td>Vivado2018.1</td>
<td>327 (88%)</td>
<td>45 (12%)</td>
<td>527227</td>
</tr>
</tbody>
</table>
Integrating the CNN into a placement tool can be used to adaptively improve its optimization strategy.

To be able to route highly-congested placement, a feedback from the CNN is used to control the cell inflation parameters.

The six highly-congested benchmarks are now routable.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Routing Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Wirelength</td>
</tr>
<tr>
<td>FPGA5-6</td>
<td>9900742</td>
</tr>
<tr>
<td>FPGA5-11</td>
<td>11814937</td>
</tr>
<tr>
<td>FPGA5-16</td>
<td>11858397</td>
</tr>
<tr>
<td>FPGA5-19</td>
<td>12069961</td>
</tr>
<tr>
<td>FPGA5-26</td>
<td>12035954</td>
</tr>
<tr>
<td>FPGA7-7</td>
<td>9540692</td>
</tr>
</tbody>
</table>
Outline for Reminder of Talk

Outline

- Background
- Deep Learning Framework
- Results
- Case Study
- Conclusion
Conclusions & Future Work

- A novel deep learning model for predicting FPGA routability during placement was proposed:
  - DLRout can be used at any stage during the placement
  - DLRout is capable of efficiently and accurately predicting the routability
  - DLRout achieves on average a 97% accuracy to predict the routability of a produced placement
  - DLRout can applied within any placement tool and it is architecture agnostic

- Our future work will focus on applying deep learning to further improve FPGA timing estimation and integrating it with the proposed routability model.
Thank you!

Guelph FPGA CAD Group
Website: https://fpga.socs.uoguelph.ca
Email: aalhyari@uoguelph.ca


Why Use Machine/Deep Learning for EDA?

Machine Learning has unique features:

• **Data-driven:** ML can learn from data to recognize complex patterns, insights and relationships in data

• **No explicit programming:** ML has the ability to extract knowledge and draw inferences from data

• **May assist in cutting CPU time:** ML can replace time consuming steps in FPGA CAD flow. ML can efficiently and accurately replace congestion estimation and routability prediction tools.

• **Provides guidance to the flow:** ML is able to provide an informative feedback that can be used by an adaptive placement flow to enhance its performance and reduce CPU time
Traditional FPGA CAD Flow

- Design Entry
- HDL (VHDL / Verilog)
- Synthesize
  - Technology Independent logic optimization
- Mapping
  - Technology Dependent Optimization
  - Mapped Netlist
- Packing
- Placement
- Routing
- Bit Stream
The FPGA CAD Flow

- Hardware design done by modelling system in HDL
- Synthesis: netlist generated
- Placement: Components placed on chip
- Routing: Connecting signals routed
- Bitstream generated to program FPGA
FPGA Placement: Challenges

- There are multiple resources constraints:
  - Heterogeneity: (LUT, FF, DSP, BRAM)
  - LUT sharing constraints
  - FF control-set constraints

- Multiple conflicting objectives:
  - Wirelength, Timing, Congestion, etc..

- High compile time:
  - Designs complexity
  - Millions of cells (logic blocks)
  - Runtime takes more than a day

Need to introduce Intelligence in FPGA Placement
Properly managing the interdependence between packing and placement is key to optimizing wirelength, timing, and congestion!
Complete Packing: LUT/FF $\rightarrow$ BLE $\rightarrow$ CLB

VPR

Traditional Pack-Place-Legalize:
- This technique **tends to pack** LUTs and FFs **at an early stage of the** optimization **thus may be difficult to unpack CLBs at a later stage if congestion is encountered** thus **may lead to unroutable solutions!!**
Partial Packing: LUT/FF → BLE

Place-SemiPack-Place-Legalize:

- More flexible than complete packing.
- However, Semi Packing tends to produce sub optimal solutions due to congestion encountered later in the placement stage.

RippleFPGA (2nd Place in ISPD16):
Place-Legalize:
- **Flat placement** allows LUTs and FFs to move throughout the placement flow **thus unrestricting** the solution space.
- However, may be **slow!**

GPlace3.0

Analytical Placement

• Prior approaches to placement use simulated annealing.
• Recently, more attention has been directed towards analytic placement, which scales better on large problem instances.

• Analytic placement approach
  1: Convert netlist to graph using Net model
  2: Perform pin propagation
  3: repeat
  4: solve non-linear equation system
  5: partition solution to enforce legality constraints
  6: until termination criteria satisfied
Experimental Setup

• GPlace was implemented using C, compiled using gcc (Red Hat 4.4.7-18) compiler.

• Binary executable files were provided from other teams for Ripple and UTplaceF placers

• Experiments were run on an Intel (Xeon CPU E3-1270 v5 @ 2.6 GHz) processor with 16 GB RAM.

• Placement solutions were routed using Xilinx Vivado 2015.4, with a patch applied to make Vivado compatible with the modified Bookshelf Format used by both academic placement tools.

• The Scikit Learn machine learning library for the Python programming language was used to implement the various classification models.

• Keras and Tensorflow are used to develop the deep learning frameworks.
A confusion matrix is an $N \times N$ matrix, where $N$ is the number of target labels (classes)

It shows the number of correct and incorrect predictions made by the classifier compared to the actual outcomes (target labels) in the actual data

E.g., binary classification problem (e.g., two classes 0|1 or T|F)

Accuracy: the proportion of the total number of predictions that are correct

$$\frac{TN + TP}{(TN + TP + FP + FN)}$$
Binary Classification Evaluation Metrics 2/2

1. Accuracy: $\frac{T_N + T_P}{T_N + T_P + F_P + F_N}$
2. Recall (Sensitivity): $\frac{T_P}{T_P + F_N}$
3. Precision: $\frac{T_P}{T_P + F_P}$
4. Specificity: $\frac{T_N}{T_N + F_P}$
5. F1-Score: $2 \times \frac{(\text{Precision} \times \text{Recall})}{(\text{Precision} + \text{Recall})}$
6. $M = \frac{(T_N \times T_P - F_P \times F_N)}{\sqrt{(T_P + F_P)(T_P + F_N)(T_N + F_P)(T_N + F_N)}}$
### Data Size Comparison for MLRoute & DLRoute

<table>
<thead>
<tr>
<th>Total amount of data (train + test)</th>
<th>MLRoute</th>
<th>DLRoute</th>
</tr>
</thead>
<tbody>
<tr>
<td>856</td>
<td>26551</td>
<td></td>
</tr>
</tbody>
</table>

- The data that was used to train, validate, and test DLRoute is 31x more the data of MLRoute.
<table>
<thead>
<tr>
<th>Area Targeting in Placement</th>
<th>Presented Frameworks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Congestion Estimation</td>
<td>MLCong, DLCong</td>
</tr>
<tr>
<td>Routability Prediction</td>
<td>MLRoute, DLRout</td>
</tr>
<tr>
<td>Flow Selection</td>
<td>MLSelect</td>
</tr>
</tbody>
</table>
Congestion Management (Inflation followed by Spreading):

1. Identify switches that are highly congested

2. Inflate all cells (LUTS) that belong to this switch by a certain amount

3. Use a bipartitioning Legalization technique to spread and move LUTs to neighboring regions to relieve current switch from overflow and congestion
Algorithm 1 LUT Inflation

Require: $Cong_B$ is the average congestion for the top 10% most congested switches
Require: $congThresh$ is the congestion threshold
Require: $lutCount$ is the number of LUTs

1: $S \leftarrow (C_1 + C_2) \max (congThresh, Cong_B)$
2: $lutSum \leftarrow 0$
3: for all LUTs do
4: \hspace{1em} $lutSum \leftarrow lutSum + \text{inputs}(LUT) + \text{outputs}(LUT)$
5: end for
6: $\mu_{LUT} \leftarrow \frac{lutSum}{lutCount}$
7: for all LUTs do
8: \hspace{1em} $cong \leftarrow \text{switchCongestion}(LUT)$
9: \hspace{2em} if $cong \geq 0.5$ and $cong < 0.675$ then
10: \hspace{3em} ratio $\leftarrow 0.8$
11: \hspace{2em} else if $cong \geq 0.675$ and $cong < 0.85$ then
12: \hspace{3em} ratio $\leftarrow 0.7$
13: \hspace{2em} else if $cong \geq 0.85$ and $cong < 1.025$ then
14: \hspace{3em} ratio $\leftarrow 0.6$
15: \hspace{2em} else if $cong \geq 1.025$ and $cong < 1.2$ then
16: \hspace{3em} ratio $\leftarrow 0.5$
17: \hspace{2em} else if $cong \geq 1.2$ then
18: \hspace{3em} ratio $\leftarrow 0.4$
19: end if
20: $\text{size}[LUT] = 1 + S \cdot \left(\frac{\text{inputs}(LUT) + \text{outputs}(LUT)}{\text{ratio} \cdot \mu_{LUT}} \right) - 1$
21: end for

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>lower</td>
<td>Lower bound of range of congestion values to inflate.</td>
<td>0.5</td>
</tr>
<tr>
<td>upper</td>
<td>Upper bound of range of congestion values to inflate.</td>
<td>1.2</td>
</tr>
<tr>
<td>adap6 parameter 1 ($C_1$)</td>
<td>See Algorithm 1 line 1.</td>
<td>0.36</td>
</tr>
<tr>
<td>adap6 parameter 2 ($C_2$)</td>
<td>See Algorithm 1 line 1.</td>
<td>0.28</td>
</tr>
<tr>
<td>congestion threshold</td>
<td>Congestion value above which a switch is considered congested.</td>
<td>0.65</td>
</tr>
</tbody>
</table>
GPlace3.0 : LUT Inlation

\[
density(\text{LUT}) = 1 + S\left(\frac{\text{inputs(\text{LUT})} + \text{outputs(\text{LUT})}}{\text{ratio.} \mu_{\text{LUT}}} - 1\right)
\]
Three problems in FPGA placement flow that are targeted using machine learning and deep learning:

- Congestion estimation
- Routability prediction
- Flow selection