Machine Learning Based Routing Congestion Prediction in FPGA High-Level Synthesis

Jieru Zhao¹, Tingyuan Liang¹, Sharad Sinha², Wei Zhang¹

¹Hong Kong University of Science and Technology (HKUST)
²Indian Institute of Technology Goa (IIT Goa)
Outline

• Background
• Machine-Learning Based Methodology
• Experimental Results
• Conclusion
Outline

• Background
• Machine-Learning Based Methodology
• Experimental Results
• Conclusion
Background: FPGA

Basic FPGA Architecture:
- LUT
- Flip-flop
- DSP Block
- Storage elements: BRAM

As an accelerator, FPGA has:
- High performance
- Low power
- Programmability & Flexibility
- Short time-to-market

A wide range of applications:
- Data center
- Image processing
- Communications
- ...
FPGA CAD design flow

RTL Implementation Flow

- High performance.
- High development cost.
- Error-prone and time-consuming.
- Hard to debug.
FPGA CAD design flow

High-Level Synthesis Flow

- Easier to implement.
- Fast design space exploration.
- Hard to grasp hardware implementation details.

RTL Implementation Flow

- High performance.
- High development cost.
- Error-prone and time-consuming.
- Hard to debug.

[Source from Xilinx]
In FPGA design, routing contributes a lot to delay and resource utilization.

### Issue: Routing Congestion

**HDL**
- Logic Synthesis
- Technology Mapping
- Netlist
- Place and Route (PAR)
- BitStream

**Netlist**
- Placement
- Routing

**Congested region**
- Wires are detoured
- Longer delays
- More routing resources
- Degrade performance
- Implementation failures
How to solve this issue?

• Physical design: Routability driven placement.
  • Time-consuming: invoke the router repeatedly.
  • Congestion prediction model guides the placement.
  • Machine learning techniques.
    Features: #bounding boxes, half-perimeter wirelength (HPWL), #pins …

• When the abstraction level increases…
  • Beneficial: easier to resolve congestion at C source-code level.
  • Time-consuming: run the C-to-bitstream flow.
  • Congestion prediction is required.

Question: How to predict routing congestion in HLS?
Challenges

• GUI-based “back to verilog” for each CLB after PAR.
  • Not applicable to build a dataset containing a large number of samples.
  • We need “back to C”.

→ Automate the process: congestion metrics per CLB -> operations in C code.

• Lack of physical metrics or features at a higher level of abstraction.
  • Physical features obtained during placement: #bounding boxes, HPWL, #pins …

→ Extract new informative features in HLS.

• Choice of machine learning models.
  • Linear, ANN, Decision tree...

→ Explore different kinds of machine learning models.
Motivational Example

For HLS-based designs,

- Applications are implemented with C/C++.
  - Hardware-friendly C code.
- Synthesis directives are applied to configure the design.
  - Loop unrolling
  - Loop pipelining
  - Array partitioning
  - Function inlining
  - ...
- Careful selection of directives.
Motivational Example

<table>
<thead>
<tr>
<th>Implementation</th>
<th>WNS (ns)</th>
<th>Max Freq. (MHz)</th>
<th>Latency (cycles)</th>
<th>Max Congestion (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>With directives</td>
<td>-13.643</td>
<td>42.3</td>
<td>$1.08 \times 10^6$</td>
<td>178.96</td>
</tr>
<tr>
<td>Without directives</td>
<td>-0.066</td>
<td>99.3</td>
<td>$1.73 \times 10^7$</td>
<td>58.51</td>
</tr>
</tbody>
</table>

Note: WNS denotes the worst negative slack.

Application: *Face Detection*.

By applying several directives as shown in the application code:
- Latency (cycles) ↓ 😊
- Congestion ↑ 😞
- Max frequency ↓ 😞

Time cost:
- Logic synthesis and PAR: nearly 7 hours.
- HLS: several minutes.

By early detection, we can avoid non-friendly coding style and improper directives!

Routing Congestion Maps.
Our Contributions

• To the best of our knowledge, we are the first to build a routing congestion prediction model in FPGA HLS.

• We develop an automatic tool to back trace the congestion metrics of CLBs and link with the HLS IR.

• We propose seven informative categories of features and compare three machine learning models.

• We propose effective solutions to resolve routing congestion.
Outline

• Background

• Machine-Learning Based Methodology

• Experimental Results

• Conclusion
Overview of our approach

Training Phase
- Construct the dataset for training.
  - Back tracing
  - Information collection
  - Feature extraction
- Model training

Prediction Phase
- Congestion prediction
- Congestion resolving
1. Automatic Back Tracing

High-Level Synthesis Flow
- C/C++
- Frontend Compilation
  - IR
    - Scheduling
    - Hardware Binding
  - Code Generation

RTL Implementation Flow
- HDL
  - Logic Synthesis
  - Technology Mapping
  - Netlist
  - Place and Route (PAR)
  - Congestion

Source Code
for (u = 0; u < window_size; u++)
#pragma HLS unroll
for (v = 0; v < window_size; v++)
#pragma HLS unroll
II[u][v] = II[u][v] + (II[u][v+1] - II[u][0]);

Intermediate Representation (IR)
id="1858":
%II_V_5_10_1 = phi i18 [ 0, %... ], [ %p_II_135_V, %...]
......
id="3373":
%tmp_35_5_s = sext i14 %r_V_4_5_s to i18
id="3374":
%p_II_135_V = add i18 %tmp_35_5_s, %II_V_5_1

HDL Description
assign p_II_135_V_fu = ($signed(tmp_35_5_s_fu) + $signed(II_V_5_10_1_reg));
......
assign p_II_112_V_fu = ($signed(tmp_35_4_11_fu) + $signed(II_V_4_12_1_reg));

CLB, Net of the output Pin
CLBLM_R_X69Y98, p_II_112_V_reg_58246[7].i_7_n_4
CLBLM_R_X69Y52, SI_V_24_35_1_reg_30030_reg_r_
......

CLB, Row, Column, Vertical Cong(%), Horizontal Cong(%)
CLBLM_R_X71Y149, 7, 129, 36.02, 48.33
CLBLM_R_X69Y52, 101, 172, 87.05, 39.04
......

C statements in the source code.
IR operations in LLVM IR.
RTL operations in HDL description.
Net of the output pin for each CLB.
Congestion metrics and corresponding CLB coordinates.
2. Information Collection

Each sample in the dataset contains:
• Features of each operation.
• Labels: corresponding congestion metrics.

To extract features, a graph is constructed to store the HLS-based information.
• Node: HLS IR operations.
• Node attributes: resource usage, bitwidth, delay...
• Edge: dependency among operations.
• Edge weight: the number of wires for the corresponding connection.

When constructing the graph, we also consider...

Resource sharing: merge the nodes that share the RTL module.

Function Interface: add I/O port in the graph.
### 3. Features

<table>
<thead>
<tr>
<th>Category</th>
<th>Feature Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bitwidth</td>
<td>Bitwidth of each operation.</td>
</tr>
<tr>
<td>Inter-connection</td>
<td>Fan-in and fan-out of each operator and their summation; #predecessors, #successors and the summation; The max. number of wires among all the connections to one-hop neighbors and its percentage of the total fan-in and fan-out. Corresponding features after including two-hop neighbors.</td>
</tr>
<tr>
<td>Resource (for each type)</td>
<td>Resource usage and utilization ratios of each operation; The total resource usage and utilization ratios of all the predecessors, successors and their summation; The max. resource usage and corresponding percentage among all the one-hop neighbors; Corresponding features after include two-hop neighbors.</td>
</tr>
<tr>
<td>Timing</td>
<td>Delay(ns) and latency(clock cycles) of each operation.</td>
</tr>
<tr>
<td>#Resource/ΔT&lt;sub/cs&lt;/sub&gt;</td>
<td>Resource usage and utilization ratios of predecessors/successors, divided by the subtraction of control states ΔT&lt;sub/cs&lt;/sub&gt;; Corresponding features for two-hop neighbors.</td>
</tr>
<tr>
<td>Operator Type</td>
<td>The operation type of each operator; The number of each kind of operations among one-hop neighbors.</td>
</tr>
<tr>
<td>Global Information</td>
<td>Resource usage of the top-level function (F&lt;sub&gt;top&lt;/sub&gt;), the function in which the operation is located (F&lt;sub&gt;op&lt;/sub&gt;) and the corresponding percentage of the resources of F&lt;sub&gt;top&lt;/sub&gt;; Target/estimated clock period and clock uncertainty of F&lt;sub&gt;top&lt;/sub&gt; and F&lt;sub&gt;op&lt;/sub&gt;; Memories: #words, #banks, #bits and #primitives(words<em>bits</em>banks); Multiplexers: number, resource usage, input size and bitwidth.</td>
</tr>
</tbody>
</table>
3. Features

In summary, for each operator, we consider:

• Features of the operator itself.
  • *Bitwidth, fan-in, fan-out, timing-related features, resource usage...*

• Features that reflect the global information.
  • *Total resource usage of the functions, target/estimated clock period, #mux...*

• Features that reflect the impact of neighboring operators.
  • One-hop neighbors.
  • Two-hop neighbors.
3. Features

Neighboring operators:

One-hop neighbors  Two-hop neighbors

Features that reflect the impact of neighbors:

✓ Inter-connections
✓ Resource usage
✓ Spatial distance

Features that reflect the impact of neighbors:

\[
\begin{array}{c|c|c|c|c}
\text{Resource} & \Delta T_{cs} \\
\hline
S_1 & 1 \\
S_2 & 3 \\
\end{array}
\]

$S_1$ and $S_2$ are two successors of operator a.

- $S_1$: execute immediately after a;
- $S_2$: execute several cycles later.

The distance constraints between a and $S_1$, $S_2$ are different.

For $S_1$, $\Delta T_{cs} = 1$.
For $S_2$, $\Delta T_{cs} = 3$.  

18
4. Model Training

Sample filtering: filter outliers and improve the quality of the dataset.

Unroll a loop with a large unrolling factor.

Multiple copies of the same operation.

These copies can be placed to distant locations.

Some are placed around the margin of FPGA.

Their congestion metrics deviate from most of the replicas (~3% outliers).

Distribution of the vertical routing congestion metrics for Face Detection on FPGA.
4. Model Training

**Machine learning models:** compare and select the best model for our problem.

- Lasso linear model.
  - Linear relationships.
  - Tuning parameter: the constant $\alpha$ that multiplies the L1-norm.

- Artificial neural network (ANN).
  - Several hidden layers between the input and output layers.
  - Tuning parameter: a number of hyperparameters.

- Gradient boosted regression trees (GBRT)
  - Multiple weak prediction models are combined to form a powerful regression ensemble.
  - Tuning parameter: the number of estimators, the learning rate...
• Based on the trained model, the congested regions can be detected in the C/C++ source code.
• Analyze the reasons of the routing congestion and find solutions.
  • Modify source code.
  • Change synthesis directives.
• Recursively mitigate congestion and optimize the target application.
Outline

• Background
• Machine-Learning Based Methodology
• Experimental Results
• Conclusion
Benchmark suite: Rosetta.
Device: Xilinx FPGA xc7z020clg484.
Our dataset contains 8111 samples.
• 302 features in seven categories.
• 3 labels: vertical congestion metric, horizontal congestion metric and their average.
Tools: Vivado design suite 2018.1, Scikit-learn machine learning library.

The property of the benchmarks are shown in:

<table>
<thead>
<tr>
<th>Congestion Metrics</th>
<th>WNS (ns)</th>
<th>Frequency (MHz)</th>
<th>Vertical Congestion (%)</th>
<th>Horizontal Congestion (%)</th>
<th>Avg. (V, H) Congestion (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>-3.253</td>
<td>75.5</td>
<td>133.33</td>
<td>178.96</td>
<td>144.87</td>
</tr>
<tr>
<td>Min</td>
<td>-13.643</td>
<td>42.3</td>
<td>5.06</td>
<td>8.90</td>
<td>6.73</td>
</tr>
<tr>
<td>Avg.</td>
<td>-8.386</td>
<td>54.4</td>
<td>60.58</td>
<td>72.47</td>
<td>64.89</td>
</tr>
</tbody>
</table>
## Estimation Accuracy

<table>
<thead>
<tr>
<th>Regression Models</th>
<th>Vertical Congestion (%)</th>
<th>Horizontal Congestion (%)</th>
<th>Avg. (V, H) Congestion (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MAE</td>
<td>MedAE</td>
<td>MAE</td>
</tr>
<tr>
<td>Not Filtering</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linear</td>
<td>13.90</td>
<td>10.88</td>
<td>18.02</td>
</tr>
<tr>
<td>ANN</td>
<td>12.19</td>
<td>7.91</td>
<td>17.68</td>
</tr>
<tr>
<td>GBRT</td>
<td>10.55</td>
<td>7.37</td>
<td>15.71</td>
</tr>
<tr>
<td>Filtering</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linear</td>
<td>12.41</td>
<td>9.20</td>
<td>17.48</td>
</tr>
<tr>
<td>ANN</td>
<td>10.23</td>
<td>7.43</td>
<td>16.61</td>
</tr>
<tr>
<td>GBRT</td>
<td>9.59</td>
<td>6.71</td>
<td>14.54</td>
</tr>
</tbody>
</table>

- 10-fold cross validation and grid search.
- Mean absolute error (MAE) measures the average value of the absolute relative errors.
- Median absolute error (MedAE) reflects the distribution of the absolute relative errors.
- Best performance: the **GBRT** model.
### Important Features

<table>
<thead>
<tr>
<th>Congestion Metrics</th>
<th>Vertical Congestion</th>
<th>Horizontal Congestion</th>
<th>Avg. (V, H) Congestion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Important Feature Categories</td>
<td>#Resource / $\Delta T_{cs}$</td>
<td>#Resource / $\Delta T_{cs}$</td>
<td>Resource</td>
</tr>
<tr>
<td>Resource</td>
<td>Resource</td>
<td>#Resource / $\Delta T_{cs}$</td>
<td></td>
</tr>
<tr>
<td>Interconnection</td>
<td>Interconnection</td>
<td>Interconnection</td>
<td></td>
</tr>
<tr>
<td>Global (Mux)</td>
<td>Global (Memory)</td>
<td>Global (Mux)</td>
<td></td>
</tr>
</tbody>
</table>

- Importance of different categories of features is assessed through the GBRT model.
- $\frac{\text{#Resource}}{\Delta T_{cs}}$ has the greatest impact on both vertical and horizontal congestion metrics.
- The related information of multiplexers and memories has a greater effect than other global features.
Case Study

Application: *Face Detection*

Baseline: Original design.

- Severe routing congestion degrades the maximum frequency significantly.
- Performance trade-off 😞

<table>
<thead>
<tr>
<th>Implementation</th>
<th>WNS (ns)</th>
<th>Max Freq. (MHz)</th>
<th>$\Delta$Latency (cycles)</th>
<th>Max Congestion Vert, Hori (%)</th>
<th>#Congested CLBs (&gt;= 100%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>-13.643</td>
<td>42.3</td>
<td>$1.08 \times 10^6$</td>
<td>133.33, 178.96</td>
<td>1272</td>
</tr>
</tbody>
</table>
### Case Study

#### Step 1: Not Inline

- The classification function contains 52 classifiers.
- *Function inlining* increases the complexity in C synthesis and generates a larger design.
- Not inline the classifiers.

<table>
<thead>
<tr>
<th>Implementation</th>
<th>WNS (ns)</th>
<th>Max Freq. (MHz)</th>
<th>∆Latency (cycles)</th>
<th>Max Congestion Vert, Hori (%)</th>
<th>#Congested CLBs (&gt;= 100%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>-13.643</td>
<td>42.3</td>
<td>$1.08 \times 10^6$</td>
<td>133.33, 178.96</td>
<td>1272</td>
</tr>
<tr>
<td>Not Inline</td>
<td>-3.504</td>
<td>74.1</td>
<td>+23</td>
<td>129.85, 97.60</td>
<td>193</td>
</tr>
</tbody>
</table>

![Baseline(H)](Baseline_H.png)  ![Baseline(V)](Baseline_V.png)  

![Step 1 Not Inline(H)](Step1_NotInline_H.png)  ![Step 1 Not Inline(V)](Step1_NotInline_V.png)
### Case Study

#### Step 2: Replication

- All the classifiers access the same completely partitioned array and multiple classifiers share the same inputs.

- A large number of interconnections.

- Modify the source code by replicating the values of the input data and sending copies to different classifiers.

<table>
<thead>
<tr>
<th>Implementation</th>
<th>WNS (ns)</th>
<th>Max Freq. (MHz)</th>
<th>$\Delta$Latency (cycles)</th>
<th>Max Congestion Vert, Hori (%)</th>
<th>#Congested CLBs (&gt;= 100%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>-13.643</td>
<td>42.3</td>
<td>$1.08 \times 10^6$</td>
<td>133.33, 178.96</td>
<td>1272</td>
</tr>
<tr>
<td>Not Inline</td>
<td>-3.504</td>
<td>74.1</td>
<td>+23</td>
<td>129.85, 97.60</td>
<td>193</td>
</tr>
<tr>
<td>Replication</td>
<td>-0.767</td>
<td>92.9</td>
<td>+0</td>
<td>106.15, 104.73</td>
<td>17</td>
</tr>
</tbody>
</table>

![Baseline(H)](image1) ![Step 1 Not Inline(H)](image2) ![Step 2 Replication(H)](image3)

![Baseline(V)](image4) ![Step 1 Not Inline(V)](image5) ![Step 2 Replication(V)](image6)
Outline

• Background
• Machine-Learning Based Methodology
• Experimental Results
• Conclusion
Conclusion

• We propose a novel machine-learning based methodology to predict routing congestion in FPGA HLS.

• Experiments show that the GBRT model achieves the highest prediction accuracy.

• Based on our model, routing congestion can be mitigated step by step and the performance of *Face Detection* can be improved significantly.
Thank you for listening!

Q & A