AutoBridge: Coupling Coarse-Grained Floorplanning with Pipelining for High-Frequency HLS Design on Multi-Die FPGAs

Licheng Guo¹, Yuze Chi¹, Jie Wang¹, Jason Lau¹, Weikang Qiao¹, Ecenur Ustun², Zhiru Zhang², Jason Cong¹

University of California Los Angeles¹, Cornell University²
lguo@ucla.edu
https://github.com/Licheng-Guo/AutoBridge
Problem

- HLS designs often suffer from low frequency
- Hard to fix the problem

```c
void kernel(
    float *dram_port0,
    float *result)
{
    .......
}
```

```verilog
module kernel()
begin
    wire dram_M_AXI_AVALID
    wire result_S_AXI_AR
    ....
end
```

My beautiful C++

Machine-generated RTL
Hard to read...

WARNING: failed to reach timing target

ERROR: routing failed

???
Reason 1: Abstraction Gap

- HLS has no physical layout information
  - How far will these two registers be apart?
  - How congested will the area be?
- Current HLS relies on inaccurate pre-characterized delay models
Reason 1: Abstraction Gap

- HLS has no physical layout information
  - How far will these two registers be apart?
  - How congested will the area be?
- Current HLS relies on inaccurate pre-characterized delay models

```c++
void top() {
    temp = foo(...);
    bar(temp, ...);
}
```

**Source C++ code**

HLS registers the connection once
(which looks reasonable)
Reason 1: Abstraction Gap

- HLS has no physical layout information
  - How far will these two registers be apart?
  - How congested will the area be?
- Current HLS relies on inaccurate pre-characterized delay models

```c
void top() {
  temp = foo(...);
  bar(temp, ...);
}
```

Source C++ code

- HLS registers the connection once (which looks reasonable)
- This is possible (and common!)
## Reason 2: FPGA Complexity

- FPGAs are increasingly large
- Multiple dies integrated together
- High delay penalty for die-crossing
  - $\sim 1$ns [Pereira FPGA’14]
- Large IPs with pre-determined location
Reason 2: FPGA Complexity

- FPGAs are increasingly large
- Multiple dies integrated together
- High delay penalty for die-crossing
  - ~1ns [Pereira FPGA’14]
- Large IPs with pre-determined location
Reason 2: FPGA Complexity

- FPGAs are increasingly large
- Multiple dies integrated together
- High delay penalty for die-crossing
  - ~1ns [Pereira FPGA’14]
- Large IPs with pre-determined location
Reason 2: FPGA Complexity

- FPGAs are increasingly large
- Multiple dies integrated together
- High delay penalty for die-crossing
  - $\sim 1\text{ns}$ [Pereira FPGA’14]
- Large IPs with pre-determined location
Reason 2: FPGA Complexity

- FPGAs are increasingly large
- Multiple dies integrated together
- High delay penalty for die-crossing
  - \( \approx 1\text{ns} \) [Pereira-2014]
- Large IPs with pre-determined location
Reason 2: FPGA Complexity

- HLS has limited consideration of those physical barriers
Reason 2:

- HLS has limited consideration of those physical barriers
- Placer often needs to pack things together to reduce die crossing
  - Increase local congestion instead
Reason 2:

- HLS has limited consideration of those physical barriers
- Placer often needs to pack things together to reduce die crossing
  - Increase local congestion instead
- Sub-optimal choice of crossing wires by the placer / router
Opportunities and Challenges

- HLS has the freedom to alter the scheduling solution
  - Potentially add more pipelining
- But where and how many?
- Will performance (cycle count) be affected?
- Scalability of the method?

```c
void top() {
    temp = foo(...);
    bar(temp, ...);
}
```
Previous Attempts

- Existing efforts focus on fine-grained delay model calibration
  - [Zheng-FPGA’12] Iteratively place & route to calibrate delay information for HLS
  - [Cong-2004] Placement-driven scheduling and binding
Previous Attempts

- Existing efforts focus on fine-grained delay model calibration
  - [Zheng-FPGA’12] Iteratively place & route to calibrate delay information for HLS
  - [Cong-2004] Placement-driven scheduling and binding
- **Not scalable**, limited to tiny designs (only ~1000s of LUTs)
  - Our benchmarks can be 100X larger and many take days to implement
Previous Attempts

- Existing efforts focus on fine-grained delay model calibration
  - [Zheng-FPGA’12] Iteratively place & route to calibrate delay information for HLS
  - [Cong-2004] Placement-driven scheduling and binding
- Not scalable, limited to tiny designs (only ~1000s of LUTs)
  - Our benchmarks can be 100X larger and many take days to implement
- Placer and router may not behave as expected
Core Idea

- Floorplan the design during HLS compilation
  - In a coarse granularity
- Add additional pipelining based on floorplan results
  - Guarantee no loss of performance
Core Idea

- Floorplan the design during HLS compilation
  - In a coarse granularity
- Add additional pipelining based on floorplan results
  - Guarantee no loss of performance
Core Idea

- Floorplan the design during HLS compilation
  - In a coarse granularity
- Add additional pipelining based on floorplan results
  - Guarantee no loss of performance
Core Idea

- Floorplan the design during HLS compilation
  - In a coarse granularity
- Add additional pipelining based on floorplan results
  - Guarantee no loss of performance
Framework Overview

Source Code

HLS Scheduling & Binding

Coarse-Grained Floorplanning

Floorplan-Aware Pipelining

RTL Generation

Constraint Generation

AutoBridge

Optimized RTL

Floorplanning Constraints

Synthesis, Placement, Routing

---

**Integrate Top-Down Physical Planning with HLS**

*Initial State*

The initial cell representing the FPGA device is divided into two child cells.

*Iteration 1*

Each cell is divided; $r_0$ divided into $r_{00}$, $r_{01}$; $r_1$ into $r_{10}$, $r_{11}$.

*Iteration 2*

*Iteration 3*

Eventually form a 2x4 grid of cells.

---

**Pipelining with Min. Area and Lossless Throughput**

1 unit of latency

$S_1 \geq S_2 + 1$

$S_2 \geq S_4$

$S_1 \geq S_3$

$S_3 \geq S_4$

$S_0 \geq S_1$

$\min. \ (S_2 - S_4) + (S_1 - S_2) + (S_1 - S_3) + 2 \ (S_3 - S_4) + (S_0 - S_1)$

---

The initial cell is divided into two child cells.

---
Framework Overview

- HLS Scheduling & Binding
- Coarse-Grained Floorplanning
- Floorplan-Aware Pipelining
- RTL Generation
- Constraint Generation
- Optimized RTL
- Floorplanning Constraints
- Synthesis, Placement, Routing

Coarse-Grained Floorplanning

- Divide the FPGA into a grid of slots
- Assign each HLS function to one slot
Coarse-Grained Floorplanning

- Divide the FPGA into a grid of slots
- Assign each HLS function to one slot
- Limit the resource utilization in each slot

limit resource usage (e.g., 70%)
Coarse-Grained Floorplanning

- Divide the FPGA into a grid of slots
- Assign each HLS function to one slot
- Limit the resource utilization in each slot
- Minimize the count of crossing-boundary wires
Coarse-Grained Floorplanning

- Divide the FPGA into a grid of slots
- Assign each HLS function to one slot
- Limit the resource utilization in each slot
- Minimize the count of crossing-boundary wires
- It is OK to have ultra-long connections
  - Will be pipelined later
Coarse-Grained Floorplanning

- Divide the FPGA into a grid of slots
- Assign each HLS function to one slot
- Use ILP to iteratively partition the design

# variables == # HLS functions
# constraints == # connections
# items in goal == # connections
Usual runtime < 10s
Coarse-Grained Floorplanning

- Divide the FPGA into a grid of slots
- Assign each HLS function to one slot
- Use ILP to iteratively partition the design

![Diagram showing initial state and iteration 1]

The initial cell representing the FPGA device is divided into two child cells.

# variables == # HLS functions
# constraints == # connections
# items in goal == # connections
Usual runtime < 10s
Coarse-Grained Floorplanning

- Divide the FPGA into a grid of slots
- Assign each HLS function to one slot
- Use ILP to iteratively partition the design

Initial State

Iteration 1

Iteration 2

The initial cell representing the FPGA device is divided into two child cells. Each cell is divided; $r_0$ divided into $r_{00}$, $r_{01}$; $r_1$ into $r_{10}$, $r_{11}$

# variables == # HLS functions
# constraints == # connections
# items in goal == # connections
Usual runtime < 10s
Coarse-Grained Floorplanning

- Divide the FPGA into a grid of slots
- Assign each HLS function to one slot
- Use ILP to iteratively partition the design
- Pipeline the cross-slot connections

The initial cell representing the FPGA device

The initial cell is divided into two child cells.

Each cell is divided; $r_0$ divided into $r_{00}$, $r_{01}$, $r_1$ into $r_{10}$, $r_{11}$

Eventually form a 2x4 grid of cells
Framework Overview
Pipeline Data Transfer Logic

- We focus on flow-control interfaces (e.g., FIFO, AXI)
- Assume a dataflow programming model
- Can be extended to non-flow-control interface
  - Refer to our paper for details
Address the Performance Concern

- Focus on when modules communicate through FIFOs
  - Hard to statically analyze the impact of additional latency
  - The additional latency may cause throughput decrease
Address the Performance Concern

- Focus on when modules communicate through FIFOs
  - Hard to statically analyze the impact of additional latency
  - The additional latency may cause throughput decrease

Note that each FIFO is being accessed by an arbitrary function
⇒ Different from simplified model such as the Synchronous Data Flow (SDF)
Address the Performance Concern

- Focus on when modules communicate through FIFOs
  - Hard to statically analyze the impact of additional latency
  - The additional latency may cause throughput decrease
- Adapt *cut-set pipelining*
  - Add the same latency to all edges in a cut
  - Equivalent to balancing the latency of reconvergent paths

![Diagram](pipeline_inter-slot_connections.png)
Address the Performance Concern

- Focus on when modules communicate through FIFOs
  - Hard to statically analyze the impact of additional latency
  - The additional latency may cause throughput decrease
- Adapt *cut-set pipelining*
  - Add the same latency to all edges in a cut
  - Equivalent to balancing the latency of reconvergent paths

![Diagram showing pipeline inter-slot connections and balance the latency of all paths.](image)
Address the Performance Concern

- Focus on when modules communicate through FIFOs
  - Hard to statically analyze the impact of additional latency
  - The additional latency may cause throughput decrease
- Adapt *cut-set pipelining*
  - Add the same latency to all edges in a cut
  - Equivalent to balancing the latency of reconvergent paths
Address the Performance Concern

- Focus on when modules communicate through FIFOs
  - Hard to statically analyze the impact of additional latency
  - The additional latency may cause throughput decrease
- Adapt *cut-set pipelining*
  - Add the same latency to all edges in a cut
  - Equivalent to balancing the latency of reconvergent paths

How to minimize area overhead?
Latency Balancing with Minimal Area Overhead

Problem: balance the latency of every pair of reconvergent paths with min area.
Latency Balancing with Minimal Area Overhead

Problem: balance the latency of every pair of reconvergent paths with min area.

- Assign variable $S_v$ for each vertex $v$
  - Analogous to the “arrival time” in static timing analysis
  - $(S_x - S_y)$ represents the latency of all path between vertex $x$ and $y$
Latency Balancing with Minimal Area Overhead

Problem: balance the latency of every pair of reconvergent paths with min area.

- Assign variable $S_v$ for each vertex $v$
  - Analogous to the “arrival time” in static timing analysis
  - $(S_x - S_y)$ represents the latency of all path between vertex $x$ and $y$
- For an edge $e_{uv}$, $(S_u - S_v)$ is no less than the additional latency needed for this edge
Latency Balancing with Minimal Area Overhead

Problem: balance the latency of every pair of reconvergent paths with min area.

- Assign variable $S_v$ for each vertex $v$
  - Analogous to the “arrival time” in static timing analysis
  - $(S_x - S_y)$ represents the latency of all path between vertex $x$ and $y$
- For an edge $e_{uv}$, $(S_u - S_v)$ is no less than the additional latency needed for this edge
- Minimize the area overhead
Latency Balancing with Minimal Area Overhead

Problem: balance the latency of every pair of reconvergent paths with min area.

- Assign variable $S_v$ for each vertex $v$
  - Analogous to the “arrival time” in static timing analysis
  - $(S_x - S_y)$ represents the latency of all path between vertex $x$ and $y$

- For an edge $e_{uv}$, $(S_u - S_v)$ is no less than the additional latency needed for this edge

- Minimize the area overhead

\[
\begin{align*}
S_1 &\geq S_2 + 1 \\
S_2 &\geq S_4 \\
S_1 &\geq S_3 \\
S_3 &\geq S_4 \\
S_0 &\geq S_1 \\
\text{min.} &\quad (S_2 - S_4) + (S_1 - S_2) + (S_1 - S_3) + 2(S_3 - S_4) + (S_0 - S_1)
\end{align*}
\]
Latency Balancing with Minimal Area Overhead

Problem: balance the latency of every pair of reconvergent paths with min area.

- Assign variable $S_v$ for each vertex $v$
  - Analogous to the “arrival time” in static timing analysis
  - $(S_x - S_y)$ represents the latency of all path between vertex $x$ and $y$
- For an edge $e_{uv}$, $(S_u - S_v)$ is no less than the additional latency needed for this edge
- Minimize the area overhead

$$S_1 \geq S_2 + 1$$
Latency Balancing with Minimal Area Overhead

Problem: balance the latency of every pair of reconvergent paths with min area.

- Assign variable $S_v$ for each vertex $v$
  - Analogous to the “arrival time” in static timing analysis
  - $(S_x - S_y)$ represents the latency of all path between vertex $x$ and $y$
- For an edge $e_{uv}$, $(S_u - S_v)$ is no less than the additional latency needed for this edge
- Minimize the area overhead

\[
\begin{align*}
S_1 & \geq S_2 + 1 \\
S_2 & \geq S_4
\end{align*}
\]
Latency Balancing with Minimal Area Overhead

Problem: balance the latency of every pair of reconvergent paths with min area.

- Assign variable $S_v$ for each vertex $v$
  - Analogous to the “arrival time” in static timing analysis
  - $(S_x - S_y)$ represents the latency of all path between vertex $x$ and $y$
- For an edge $e_{uv}$, $(S_u - S_v)$ is no less than the additional latency needed for this edge
- Minimize the area overhead

\[
S_1 \geq S_2 + 1 \\
S_2 \geq S_4 \\
S_1 \geq S_3
\]
Latency Balancing with Minimal Area Overhead

Problem: balance the latency of every pair of reconvergent paths with min area.

- Assign variable $S_v$ for each vertex $v$
  - Analogous to the “arrival time” in static timing analysis
  - $(S_x - S_y)$ represents the latency of all path between vertex $x$ and $y$
- For an edge $e_{uv}$, $(S_u - S_v)$ is no less than the additional latency needed for this edge
- Minimize the area overhead

![Diagram of latency balancing with minimal area overhead](image)

- $S_1 \geq S_2 + 1$
- $S_2 \geq S_4$
- $S_1 \geq S_3$
- $S_3 \geq S_4$
Latency Balancing with Minimal Area Overhead

Problem: balance the latency of every pair of reconvergent paths with min area.

- Assign variable $S_v$ for each vertex $v$
  - Analogous to the “arrival time” in static timing analysis
  - $(S_x - S_y)$ represents the latency of all path between vertex $x$ and $y$
- For an edge $e_{uv}$, $(S_u - S_v)$ is no less than the additional latency needed for this edge
- Minimize the area overhead

1 unit of latency

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>w=1</td>
<td>w=1</td>
<td>w=1</td>
<td>w=1</td>
<td>w=2</td>
</tr>
</tbody>
</table>

$S_1 \geq S_2 + 1$
$S_2 \geq S_4$
$S_1 \geq S_3$
$S_3 \geq S_4$
$S_0 \geq S_1$
Latency Balancing with Minimal Area Overhead

Problem: balance the latency of every pair of reconvergent paths with min area.

- Assign variable $S_v$ for each vertex $v$
  - Analogous to the “arrival time” in static timing analysis
  - $(S_x - S_y)$ represents the latency of all path between vertex $x$ and $y$
- For an edge $e_{uv}$, $(S_u - S_v)$ is no less than the additional latency needed for this edge
- Minimize the area overhead

1 unit of latency

$w=1$ $w=1$ $w=1$ $w=2$

$S_1 \geq S_2 + 1$
$S_2 \geq S_4$
$S_1 \geq S_3$
$S_3 \geq S_4$
$S_0 \geq S_1$

\[
\text{min. } (S_2 - S_4) + (S_1 - S_2) + (S_1 - S_3) + 2(S_3 - S_4) + (S_0 - S_1)
\]
Latency Balancing with Minimal Area Overhead

Problem: balance the latency of every pair of reconvergent paths with min area.

- Assign variable $S_v$ for each vertex $v$
  - Analogous to the “arrival time” in static timing analysis
  - $(S_x - S_y)$ represents the latency of all path between vertex $x$ and $y$
- For an edge $e_{uv}$, $(S_u - S_v)$ is no less than the additional latency needed for this edge
- Minimize the area overhead

System of Difference Constraints (Polynomial Time Solvable)

\[
\begin{align*}
S_1 &\geq S_2 + 1 \\
S_2 &\geq S_4 \\
S_1 &\geq S_3 \\
S_3 &\geq S_4 \\
S_0 &\geq S_1
\end{align*}
\]

\[
\text{min. } (S_2 - S_4) + (S_1 - S_2) + (S_1 - S_3) + 2(S_3 - S_4) + (S_0 - S_1)
\]
Benchmarks

- A total of 43 design configurations
- 16 of them originally failed in routing
- From 147 MHz to 297 MHz on average (~2X)
- Negligible difference in resource utilization or cycle count.
Case Study 1

- Stencil Computation, 16 configurations
  - Opt: avg 266 MHz (3.1X)
  - Opt: avg. 273 MHz (3.9X)

- Difference in Resource Utilization
  - LUT: -0.26%
  - FF: +0.78%
  - BRAM: +4.68%
  - DSP: +0.00%

Comparison of the 4-PE Design on U280
Case Study 2

- Gaussian Elimination, 8 configurations
  - Opt: avg. 334 MHz (1.4X)
  - Opt: avg. 335 MHz (1.5X)

- Difference in Resource Utilization
  - LUT: -0.14%
  - FF: -0.04%
  - BRAM: -0.03%
  - DSP: +0.00%

Default: avg. 245 MHz

Default: avg. 223 MHz

Comparison of the 24x24 Design on U250
Case Study 3

- CNN Accelerator, 14 configurations

**Opt: avg. 316 MHz (2.3X)**

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Fmax (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>13x2</td>
<td>13x4</td>
</tr>
<tr>
<td>13x6</td>
<td>13x10</td>
</tr>
<tr>
<td>13x12</td>
<td>13x14</td>
</tr>
<tr>
<td>13x16</td>
<td></td>
</tr>
</tbody>
</table>

**Opt: avg. 328 MHz (1.5X)**

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Fmax (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>13x2</td>
<td>13x4</td>
</tr>
<tr>
<td>13x6</td>
<td>13x10</td>
</tr>
<tr>
<td>13x12</td>
<td>13x14</td>
</tr>
<tr>
<td>13x16</td>
<td></td>
</tr>
</tbody>
</table>

- Difference in Resource Utilization
  - LUT: -0.08%
  - FF: -0.16%
  - BRAM: -0.02%
  - DSP: +0.00%

Comparison of the 16x13 Design on U250

- Default: avg. 140 MHz
- Opt: avg. 316 MHz (2.3X)
- Opt: avg. 328 MHz (1.5X)
Impact of Pipelining and Floorplanning

- Is it possible that only one of them is the key factor?
  - Baseline: (-) floorplanning, 8 slots (-) pipelining
  - AutoBridge: (+) floorplanning, 8 slots (-) pipelining
  - Case 1: (-) floorplanning (+) pipelining
  - Case 2: (+) floorplanning, 4 slots (neglect the DDRs) (-) pipelining

Control Experiments Based on Systolic Arrays on U250
Projects Using AutoBridge

- AutoSA: Polyhedral-Based Systolic Array Auto-Compilation
  - [https://github.com/UCLA-VAST/AutoSA](https://github.com/UCLA-VAST/AutoSA)
- TAPA: Extending High-Level Synthesis for Task-Parallel Programs
  - [https://github.com/Blaok/tapa](https://github.com/Blaok/tapa)
- Acceleration of Bayesian Network Inference (in submission)
- Acceleration of Single-Source-Shortest-Path algorithm (in submission)
Projects Using AutoBridge

- AutoSA: Polyhedral-Based Systolic Array Auto-Compilation
  - https://github.com/UCLA-VAST/AutoSA
- TAPA: Extending High-Level Synthesis for Task-Parallel Programs
  - https://github.com/Blaok/tapa
- Acceleration of Bayesian Network Inference (in submission)
- Acceleration of Single-Source-Shortest-Path algorithm (in submission)
Thank You!