LiveHD: A Multi-threaded Fast Hardware Compiler for HDLs

Sheng-Hong Wang, Sakshi Garg, and Jose Renau

Department of CSE
University of California - Santa Cruz

Outline

- Research Problem and Background
- LiveHD Compilation Framework
- LiveHD Next Targets
- Conclusions

Outline

- **Research Problem and Background**
- LiveHD Compilation Framework
- LiveHD Next Targets
- Conclusions

Hardware Design with New HDLs Is Still Painful

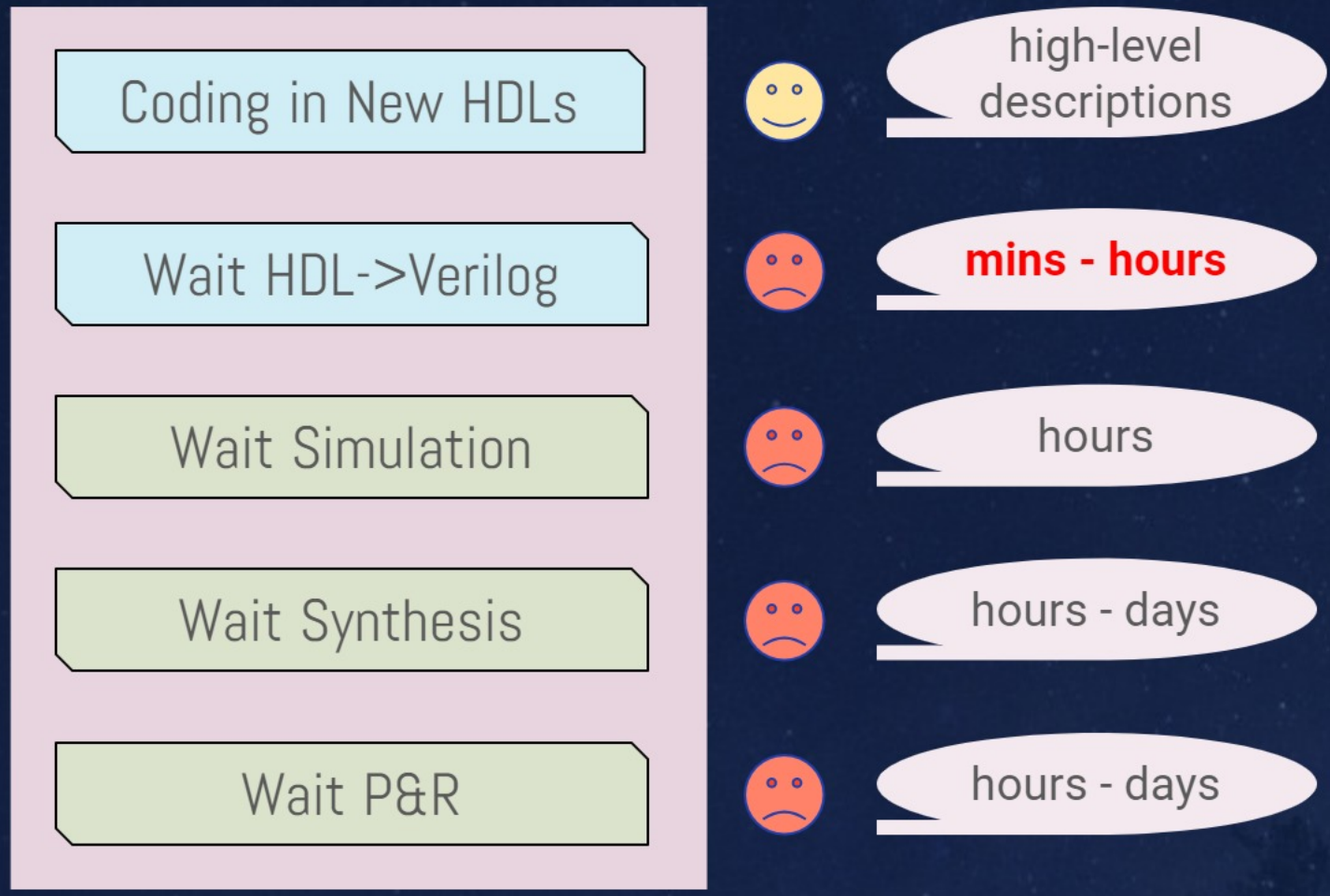
Coding in New HDLs

high-level descriptions

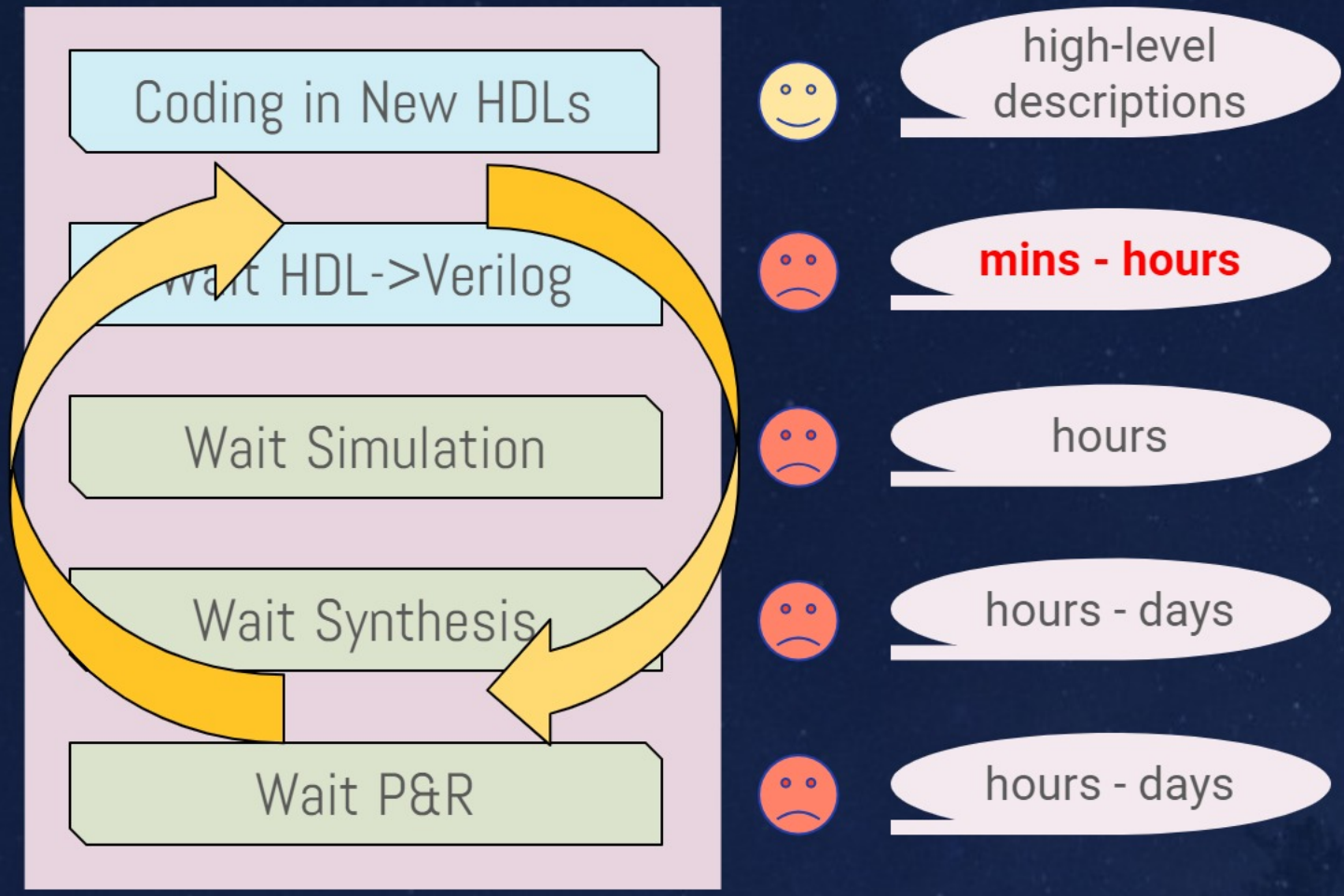
Wait HDL->Verilog

mins - hours

Hardware Design with New HDLs Is Still Painful



Hardware Design with New HDLs Is Still Painful



Live Hardware Development (LiveHD)

- Fast HDLs Compilation Infrastructure
  - LNAST IR
  - LGraph IR
- Multi-Languages support
  - Verilog
  - FIRRTL
  - Pyrope
- Parallel HDLs Compilation
  - **Current focus**
- Incremental HDLs Compilation
  - Final phase

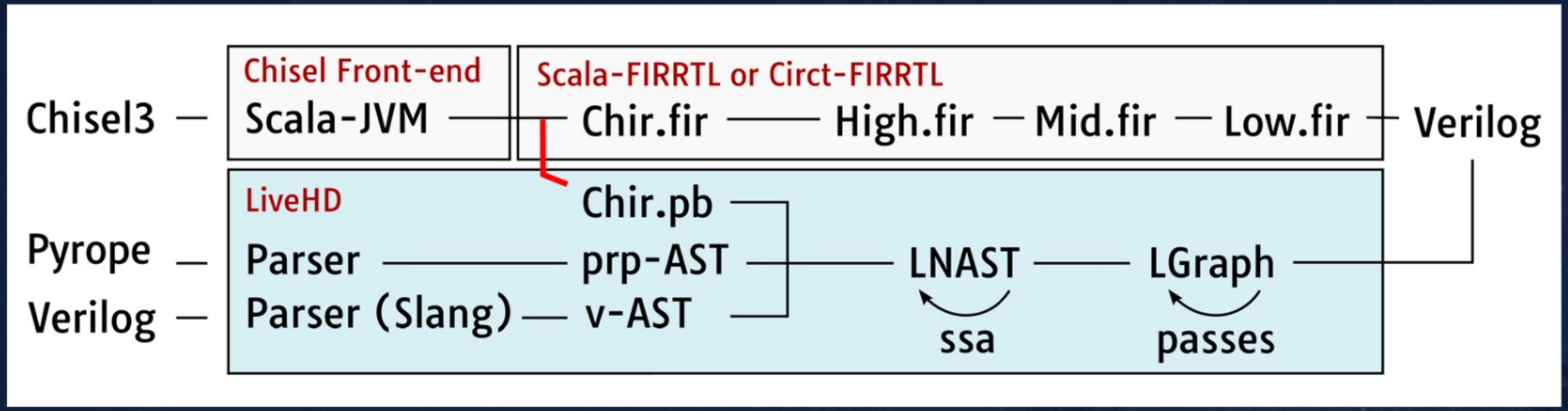
Outline

- Research Problem and Background
- **LiveHD Updates**
- LiveHD Next Targets
- Conclusions

FIRRTL Compilation Flow in LiveHD

LiveHD supports the **highest level** FIRRTL (CHIRRTL)

- Alternative FIRRTL compiler
  - GCC v.s. Clang for C++
- **33.4x faster than FIRRTL compiler (single thread)**



Parallelized LiveHD: Background

- For a pass, two modules are independent if ...
  - They don't need to communicate to each other
  - Can parallely execute the pass

- Otherwise, two modules must be processed in an order
  - caller() -> callee() or
  - callee() -> caller()

Parallelized LiveHD: Challenges

- Have to understand the modules' dependency relation for parallelism
- How do non-hardware language deal function declaration dependency?
  - pre-process phase
    - #include
    - import

```cpp
using namespace std;
#include <queue>
#include <string>
#include "foo.hpp"
#include "bar.hpp"

class TreeNode {
public:
  int value;
  TreeNode *left, *right;

  TreeNode(int x, int y) {
    value = foo(x) + bar(y);
    left = right = nullptr;
  }
};
```

Parallelized LiveHD: Challenges

HDLs don't have C++-like include syntax

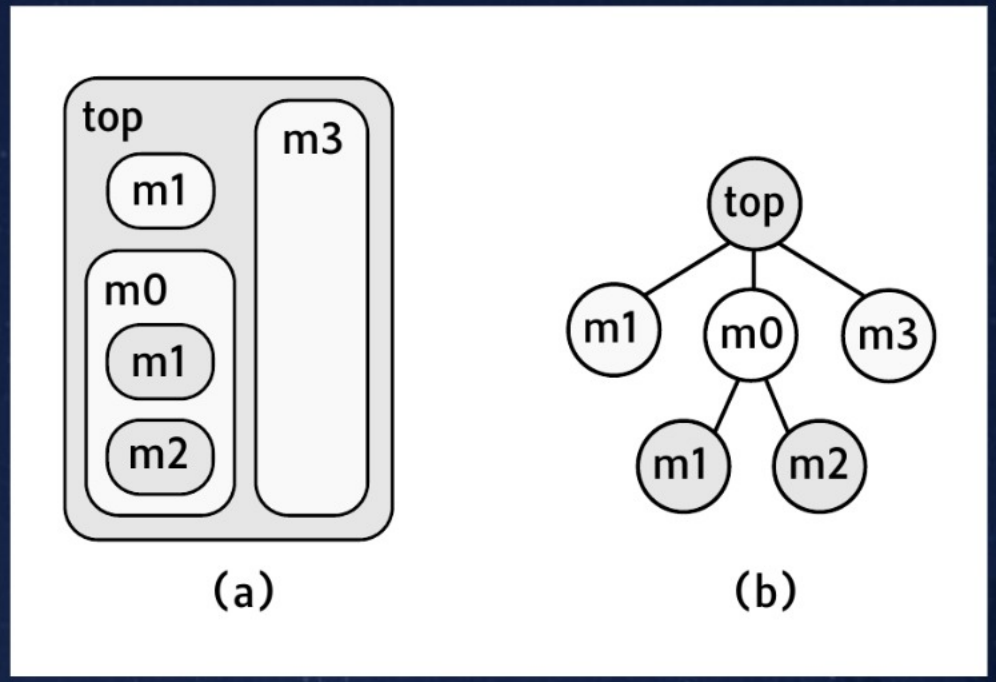
- Verilog do not require dependency for ***include***-> useless
- In HDLs, the import tends to be "implicit".

- LiveHD builds dependency tree at the same time with IR construction!

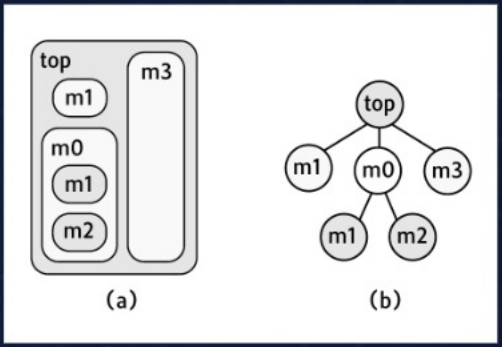
```
1  using namespace std;
2  #include <queue>
3  #include <string>
4  #include "foo.hpp"
5  #include "bar.hpp"
6
7  class TreeNode {
8  public:
9    int value;
10   TreeNode *left, *right;
11
12   TreeNode(int x, int y) {
13   | value = foo(x) + bar(y);
14   | left = right = nullptr;
15   }
16 };
```

Parallelized LiveHD: Two Mechanisms

- Full-parallelism
  - If modules are independent @ a pass
- Bottom-up parallelism
  - If modules are dependent @ a pass
  - Extract parallelism from dependency tree



Parallelized LiveHD: Flow Chart



1 Source_to_LNAST

2 LNAST_SSA

3 LNAST_to_LGraph

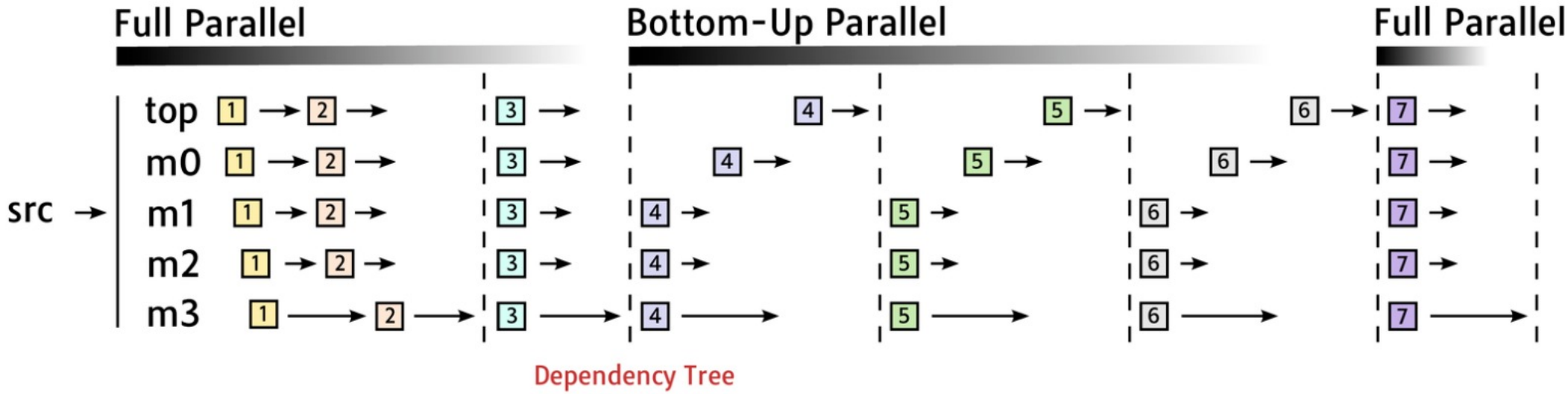
4 CDCPATI

Copy_Propagation
Dead_Code_Elimination
Constant_Propagation
Peephole_Optimization
Attribute_Resolving
Tuple_Resolving
IO_Construction

5 FIRRTL_Analysis

6 FIRRTL_Mapping + Bitwidth_Inference

7 Verilog_Generation



Parallelized LiveHD: Pass Parallelization Type

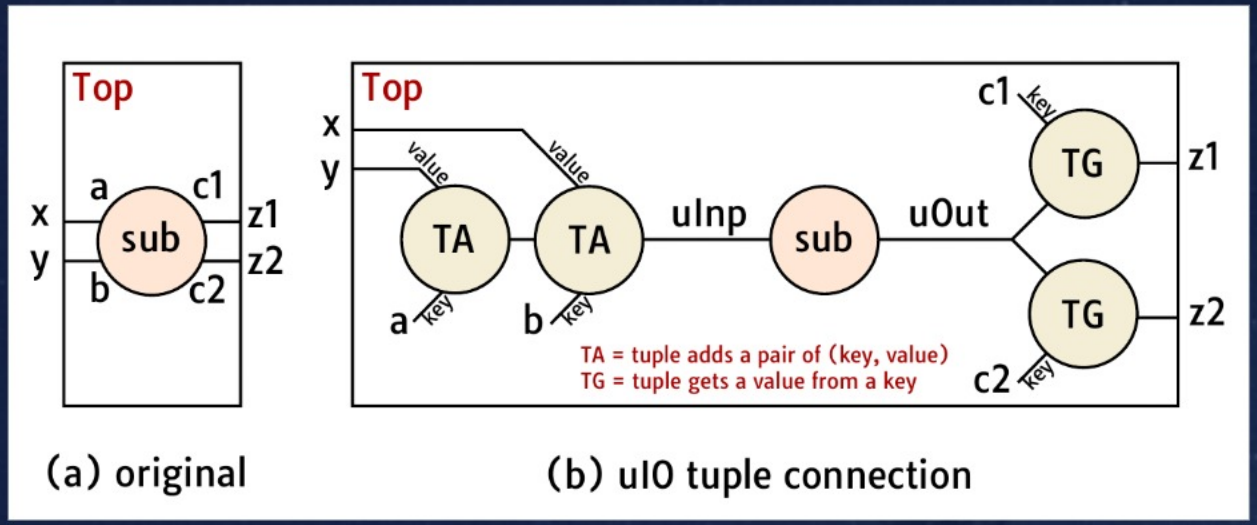
Name	Functionality	Parallelization Type	
		Full	*Bottom-up*
Source_to_LNAST[1]	Verilog to parse tree to LNAST	✓	
	Pyrope to parse tree to LNAST	✓	
	CHIRRTL protocol buffer to LNAST	✓	
LNAST_SSA	SSA transformation for LNAST	✓	
LNAST_to_LGraph	LNAST to LGraph translation	✓	
CDCPATI[2]	Copy propagation	✓	
	Dead code elimination	✓	
	Constant propagation	✓	
	Peephole optimization	✓	
	Attribute resolving		✓
	Tuple struct resolving		✓
	IO construction		✓
FIRRTL_Analysis[3]	FIRRTL operator width analysis		✓
FIRRTL_Mapping[3]	FIRRTL and LGraph operator mapping		✓
Bitwidth_Inference	Bitwidth inference and optimization		✓
Verilog_Generation	Back-end Verilog code generation	✓	

Parallelized LiveHD: Why Modules Are Dependent?

- The main reasons: ***IO! IO! IO!***
  - **IO connections** of the graph hierarchy
    - Top need to know sub's IO positions before connect
  - **IO tuple** data structure resolving
  - **IO attribute** propagation
    - Ex. FIRRTL bit analysis
    - Ex. Bitwidth optimization
- LiveHD pursue as much full-parallelism pass as possible

Parallelized LiveHD: Novel LNAST->LGraph Design

- ***uIO*** mechanism enables full-parallelim
  - Instead of sub-graph IO, connecting to pseudo ***uIO***
  - Isolate IO dependency between top and sub!
- LiveHD pursue as much full-parallelism pass as possible!



Fast LiveHD: Novel Integrated Passes

- Integrated Passes to share the same graph traversals
- Largely increase the compilation throughput!

Name	Functionality	Parallelization Type	
		Full	*Bottom-up*
Source_to_LNAST[1]	Verilog to parse tree to LNAST	✓	
	Pyrope to parse tree to LNAST	✓	
	CHIRRTL protocol buffer to LNAST	✓	
LNAST_SSA	SSA transformation for LNAST	✓	
LNAST_to_LGraph	LNAST to LGraph translation	✓	
CDCPATI[2]	Copy propagation	✓	
	Dead code elimination	✓	
	Constant propagation	✓	
	Peephole optimization	✓	
	Attribute resolving		✓
	Tuple struct resolving		✓
	IO construction		✓
FIRRTL_Analysis[3]	FIRRTL operator width analysis		✓
FIRRTL_Mapping[3]	FIRRTL and LGraph operator mapping		✓
Bitwidth_Inference	Bitwidth inference and optimization		✓
Verilog_Generation	Back-end Verilog code generation	✓	

Evaluations

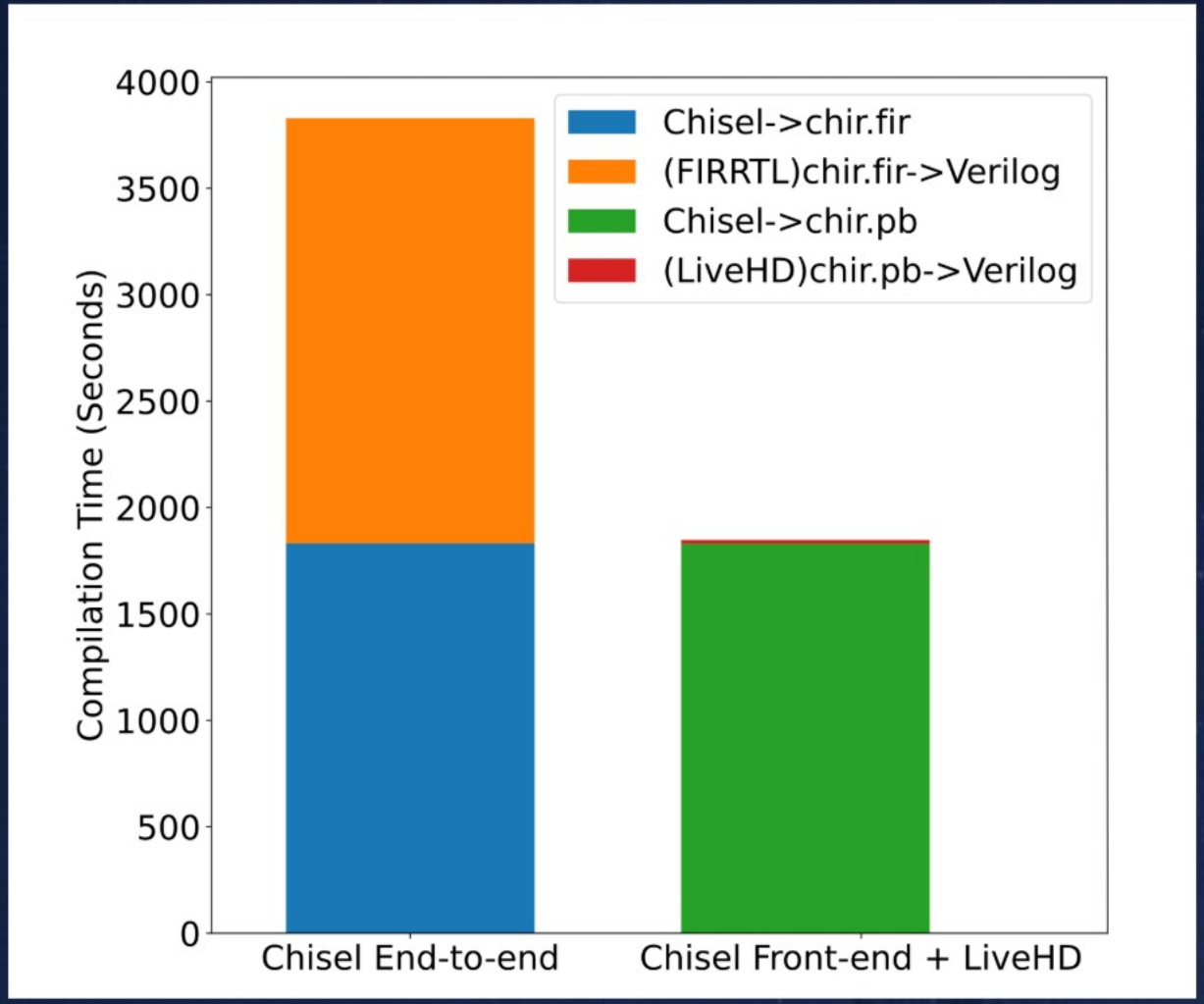
- Compile a 1.3-million loc combinational circuit
  - 3381 modules, average size = 383 loc
  - Balanced dependency tree with 4 children

Evaluations: CHIRRTL -> Verilog

- Compile a 1.3-million loc combinational circuit
  - FIRRTL Compiler: chir.fir -> Verilog
  - LiveHD Compiler: chir.pb -> Verilog

	Time	Speed-up	Scalability
FIRRTL Compiler	1998s	1x	N/A
LiveHD thread = 1	59.83s	33.4x	1x
LiveHD thread = 2	34.25s	58.33x	1.74x
LiveHD thread = 3	25.98s	76.90x	2.3x
LiveHD thread = 4	21.96s	90.98x	2.72x
LiveHD thread = 8	17.45s	114.49x	3.42x

Evaluations: Chisel Source Code -> Verilog



Outline

- Research Problem and Background
- LiveHD Compilation Framework
- **LiveHD Next Targets**
- Conclusions

LiveHD Next Target: Development

- Compile Berkeley out-of-order (BOOM) core from ch-pb
  - support memory instance

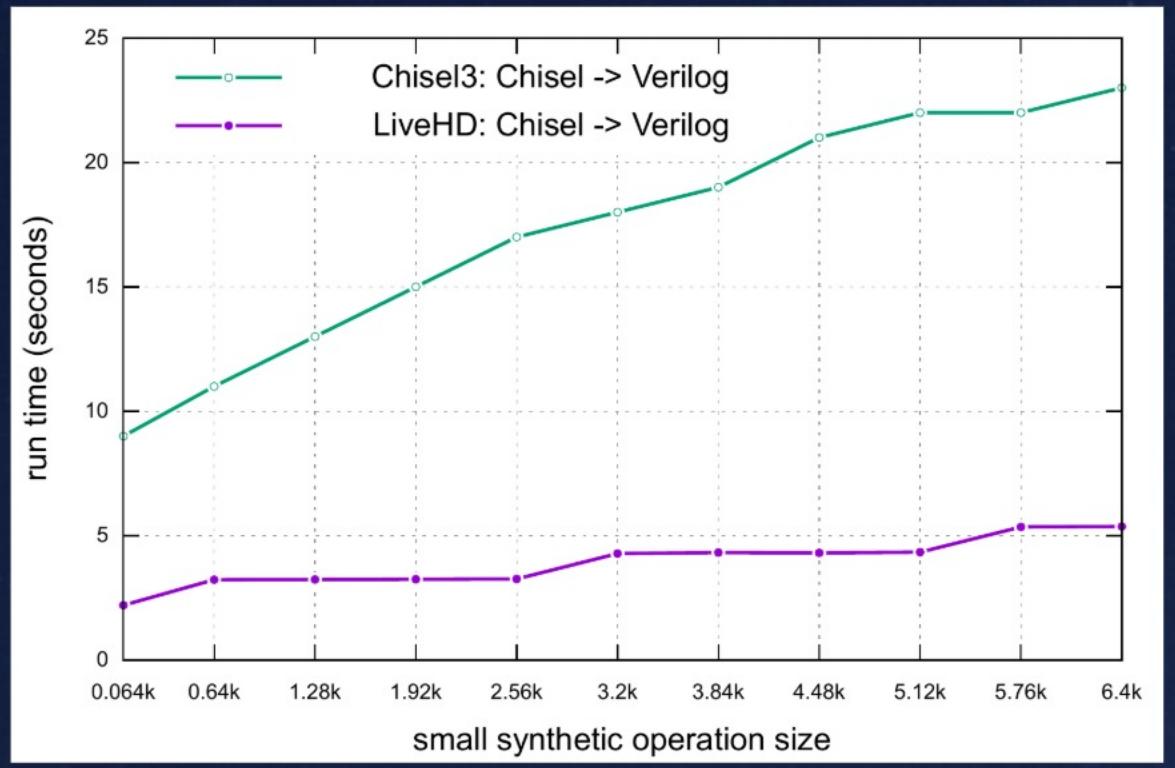
- Improve the scalability of multi-threaded LiveHD compilation

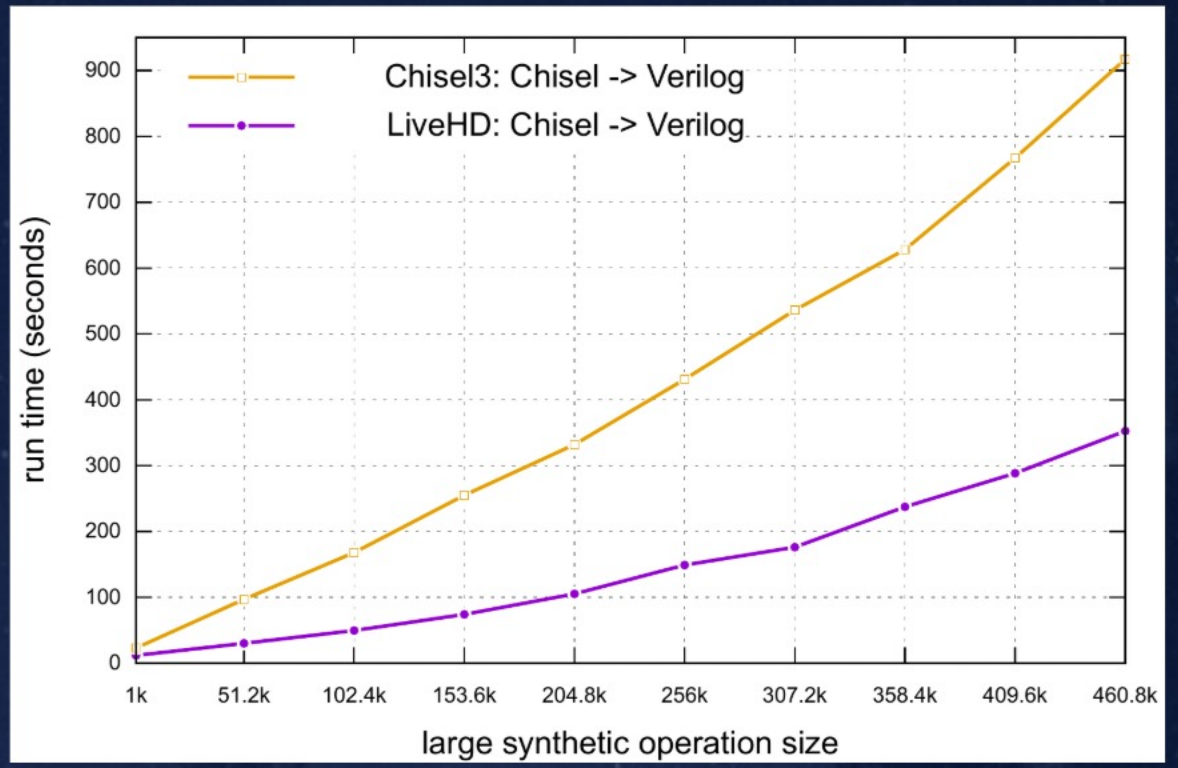
Conclusions

- LiveHD can compile highest FIRRTL semantic directly
- We kept focusing on build fast LiveHD
  - **33.4x** speed-up over FIRRTL compiler
  - **114.5x** speed-up with 8-threads compilation
- More opportunities in solving scalability issues

Appendix

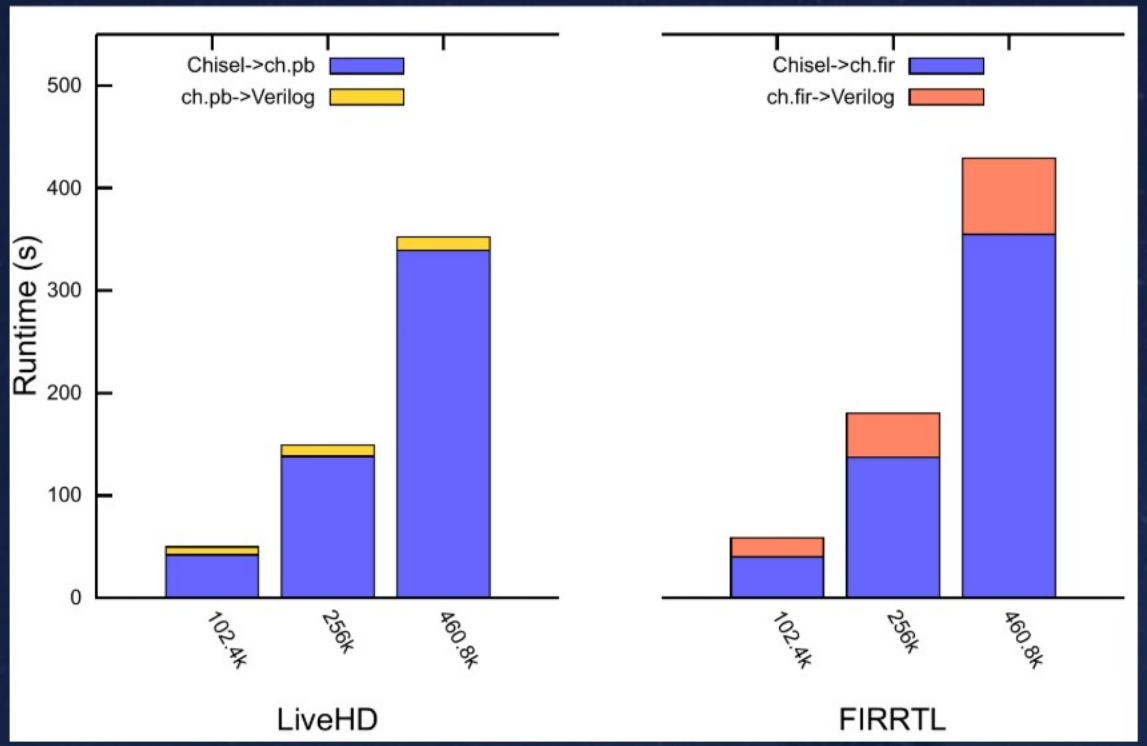
Evaluations





Evaluations

- Chisel->ch-pb/ch-fir is a giant overhead
  - the reason we don't like DSL HDL.
  - another reason for Pyrope HDL



LiveHD Accepted Publications 2021

- Design Decisions in LiveHD for HDLs Compilation, ***Sheng-Hong Wang*** and Jose Renau, 1st Workshop on Languages, Tools, and Techniques for Accelerator Design (LATTE), April 2021

- A Parallel HDL Compilation Framework, ***Sheng-Hong Wang***, Sakshi Garg, Hunter James Coffman, Kenneth Mayer, and Jose Renau, 4th Workshop on Open-Source EDA Technology (WOSET), November 2021

- Guide for Rapid Creation of New HDLs, Sakshi Garg, ***Sheng-Hong Wang***, Hunter James Coffman, and Jose Renau, 4th Workshop on Open-Source EDA Technology (WOSET), November 2021