Testing Compilers for Programmable Switches Through Switch Hardware Simulation

Michael D. Wong, Aatish Kishan Varma, Anirudh Sivaraman
Programmable Switches

- Switch programmability
  - Power
  - Cost
  - Performance
Programming Languages

Lyra: A Cross-Platform Language and Compiler for Data Plane Programming on Heterogeneous ASICs
Jiaqi Gao1, Ennan Zhai, Hongqiang Harry Liu, Rui Mao, Yu Zhou, Bingchuan Tian, Chen Sun
Dennis Cal, Ming Zhang, Minlan Yu

1Alibaba Group, 2Harvard University, 3Tsinghua University, 4Nanjing University

Packet Transactions: High-Level Programming for Line-Rate Switches
Anirudh Sivaraman, Alvin Cheung, Mihai Budiu, Changhoon Kim, Mohammad Alizadeh, Hari Balakrishnan,
George Varghese, Nick McKeown, Steve Lickteig

1MIT CSAIL, 2University of Washington, 3VMWare Research, 4Barefoot Networks, 5Microsoft Research, 6Stanford University
Building Switch Compilers

- Compiler development in general is hard
  - Need knowledge of underlying hardware architecture
  - Requires significant engineering effort

- Switch compiler development introduces new challenges
Switch Compilation Challenges

- Limited hardware resources
  - Limited pipeline stages, memory, ALUs, etc.

- Pipeline architecture
  - Computations must be mapped to pipeline stages that respect dependencies

- All-or-nothing nature
  - A program runs at line-rate if it can be mapped to the pipeline or it is rejected
Druzhba

- Low-level switch pipeline simulator
- Executes compiler-generated machine code programs
- Compiler developers observe the correctness of compiler mappings
Druzhba Machine Model

- Druzhba models the hardware details of the pipeline architecture
  - Details of arithmetic logic units (ALUs)
  - Packet header vectors (PHVs) instead of packets
  - Input and output multiplexers
Druzhba Machine Model

PHV Container

PHV Container

PHV
Druzhba Machine Model
Druzhba Machine Model
Druzhba Machine Model
Druzhba Machine Model

Pipeline Stage 1

PHV Container
PHV Container
PHV
Input Muxes

Stateless ALU
Stateless ALU
Stateful ALU
Stateful ALU

Output Muxes

PHV Container
PHV Container
PHV
Druzhba Machine Model
Druzhba Machine Model
Druzhba Machine Model
Druzhba Machine Model

Limitations: does not model matching, parsing, and scheduling
Druzhba Overview

- **dgen** pipeline generator
  - Generates simulation program to represent hardware details of the pipeline

- **dsim** pipeline simulator
  - Executes a machine code program using dgen’s generated pipeline
Hardware Specification

- dgen uses the hardware specification to generate a file with the pipeline configuration details

2 x 2
Pipeline depth x width

ALU specification

- Optimizations applied to the pipeline description to reduce overall simulation time
Hardware Simulation

- dsim simulates the hardware design specified in the pipeline description

- dsim takes in as input the machine code from the compiler
  - The machine code configures the behavior for ALUs and muxes

- A compiler is tested by fuzzing a compiler-generated program with random PHVs
Hardware Simulation

Machine code

Input PHV

Stateless ALU
Stateful ALU
State storage

Pipeline Stage 1

Output PHV

Stateless ALU
Stateful ALU
State storage

Pipeline Stage 2

Output PHV

Input Muxes

Output Muxes

PHV Container

PHV

Input PHV

PHV Container

Machine code

100 001
Compiler Testing Workflow

1. Provide transactional specification that captures intended behavior

2. Fuzz both the dsim pipeline and the transactional specification

3. Check that the dsim pipeline and the transactional specification produce the same behavior
Correctness

- Correct compiler mapping
  - The output PHVs from both the dsim pipeline and the transactional specification are equal

- Erroneous compiler mapping
  - An input PHV yields two different results from the dsim pipeline and transactional specification
Compiler Testing Example

Traffic generator

Input PHV

Transactional spec

dsim
Compiler Testing Example

Traffic generator → Transactional spec → Input PHV

Equal? → PHV → Equal → PHV → Equal → PHV → Equal → PHV → Equal → PHV

dsim → PHV → PHV → PHV → PHV → PHV
Compiler Testing Example

Traffic generator → Input PHV → Transactional spec → PHV → Equal → PHV → Equal → PHV → Equal → PHV → Equal → PHV → Equal → PHV → Equal → PHV → Correct

dsim
Compiler Testing Example
Compiler Testing Example

Traffic generator \[\rightarrow\] Input PHV

Transaction spec

PHV \[\uparrow\] PHV \[\uparrow\] PHV \[\uparrow\] PHV

NOT Equal \[\rightarrow\] Equal \[\rightarrow\] Equal \[\rightarrow\] Equal

Compilation error

dsim

PHV \[\uparrow\] PHV \[\uparrow\] PHV \[\uparrow\] PHV
Testing Chipmunk

- Druzhba tested Chipmunk, a compiler for the Domino programming language
- The compilations of 120+ Chipmunk programs were validated
Conclusion

- We can model the low-level hardware details of the switch chip
- Druzhba can simulate compiler-generated machine code programs
- Compiler developers observe correctness of compiler mappings
- Code: https://github.com/chipmunk-project/druzhba-simulator
back up slides
False Positives

```
1 void program1(struct Packet pkt) {
2     if (filter1 != 0 && filter2 != 0 && filter3 != 0) {
3         pkt.member = 1;
4     }
5     filter1 = 1;
6     filter2 = 1;
7     filter3 = 1;
8 }
```

```
1 void program2(struct Packet pkt) {
2     if (filter1 != 0 && filter2 != 0 && filter3 != 0) {
3         pkt.member = 1;
4         filter1 = 1;
5         filter2 = 1;
6     } else {
7         filter1 = 1;
8         filter2 = 1;
9     }
10     filter3 = 1;
11 }
```
False Positives

Program 1

Program 2

Compiler

Does the program fit?

Yes

No
False Positives

Program 1

Program 2

Compiler

Does the program fit?

Yes

Execute on switch

No
False Positives

Program 1

Compiler

Does the program fit?

Yes

Execute on switch

No

Rejected

Program 2
Switch Architecture
Compilation to pipeline

```python
if (count == 10):
    count = 0
    pkt.field = 1
else:
    count++
    pkt.field = 0
```
Hardware Specification

- Pipeline dimensions (depth, width)
  - Number of total stages and number of ALUs per stage

- ALU specifications
  - ALU DSL used to specify capabilities of switch ALUs

\[ 2 \times 2 \]

Pipeline depth x width  ALU specification
Optimizations

- Sparse conditional constant propagation
  - Constant propagation of machine code values followed by interpretation of control flow

- Function inlining
  - Replace pipeline description
Compiler Testing Workflow

- Hardware spec
- Program
- Compiler
- Machine code
- Pipeline desc.
- dgen
- dsim