Building Compilers for Reconfigurable Switches

Lavanya Jose, Lisa Yan, Nick McKeown, and George Varghese

Research funded by AT&T, Intel, Open Networking Research Center.
In the next 20 minutes

- Fixed-function switch chips will be replaced by reconfigurable switch chips
- We will program them using languages like P4
- We need a compiler to compile P4 programs to reconfigurable switch chips.
Fixed-Function Switch Chips
Control Flow Graph

Switch Pipeline

Control Flow Graph

Parser

L2 Table

IPv4 Table

IPv6 Table

ACL Table

Fixed Action

Fixed Action

Fixed Action

Fixed Action

Queues
Fixed-Function Switch Chips Are Limited

1. Can’t add new forwarding functionality
2. Can’t add new monitoring functionality
Fixed-Function Switch Chips

Control Flow Graph
Switch Pipeline
Fixed-Function Switch Chips Are Limited

1. Can’t add new forwarding functionality
2. Can’t add new monitoring functionality
3. Can’t move resources between functions
Reconfigurable Switch Chips

Control Flow Graph

Switch Pipeline

Parser

Match Table
Action Macro

Match Table
Action Macro

Match Table
Action Macro

Match Table
Action Macro

Queues
Mapping Control Flow to Reconfigurable Chip.

Control Flow Graph

Switch Pipeline

Parser

L2 Table

L2 Action Macro

IPv4 Table

IPv4 Action Macro

IPv6 Table

IPv6 Action

ACL Table

ACL Action Macro

Queues
Reconfigurable Switch Chips

Control Flow Graph

Switch Pipeline

Parser

L2 Table
L2 Action Macro

MyEncap

IPv4
Action

v4

IPv6
Action

v6

ACL

ACL Table
ACL Action Macro

Queues
Match + Action Processor: pipelined and in-parallel
Reconfigurability: the norm in 5 years

- Reconfigurability adds mostly to logic.
- Logic is getting relatively smaller.
- The cost of reconfigurability is going down.
- Fixed switch chip area today:
  - I/O (40%), Memory (40%),
  - Wires, Logic
Fixed Function Broadcom Tomahawk: 3.2 Tbps
Reconfigurable Cavium Xpliand: 3.2 Tbps
Reconfigurable chips are inevitable.
Configuring Switch Chips

P4 code

Compiler

Compiler Target

Parser

Match Table
Action Macro

Match Table
Action Macro

Match Table
Action Macro

Match Table
Action Macro

Queues
P4 (http://p4.org/)

Parser (ANCS’13)

```
parser parse_ethernet {
extract(ethernet);
select(latest.etherType) {
  0x800 : parse_ipv4;
  0x86DD : parse_ipv6;
}
}
```

Match Action Tables

```
table ipv4_lpm {
  reads {
    ipv4.dstAddr : lpm;
  }
  actions {
    set_next_hop;
    drop;
  }
}
```

Control Flow Graph

```
control ingress {
  apply(l2_table);
  if (valid(ipv4)) {
    apply(ipv4_table);
  }
  if (valid(ipv6)) {
    apply(ipv6_table);
  }
  apply (acl);
}
```

Queues
What does reconfigurability buy us?
Benefits of Reconfigurability

- Use resources efficiently
  - Multiple tables per stage
  - Big table in multiple stages
- Use fewer stages
Naïve Mapping: Control Flow Graph

Control Flow
Switch Pipeline

L2 -> v4 -> v6 -> ACL

Parser

L2 Table
IPv4 Table
IPv6 Table
ACL Table

Action

Action

Action

Action

Queues
Table Dependency Graph (TDG)
Efficient Mapping: TDG

Control Flow Graph

Switch Pipeline

Parser

L2 Table

Action

IPv4 Table

V4 Action Macro

IPv6 Table

V6 Action Macro

ACL Table

Action

Queues
Resource constraints

Control Flow Graph
Switch Pipeline
More resource constraints

Table parallelism
Action Memory
Memory Type
Action ALU input
Header widths
The Compiler Problem

Map match action tables in a TDG to a switch pipeline while respecting dependency and resource constraints.
Step 1: P4 Program

Step 2: Control Flow Graph

Step 3: Table Dependency Graph

Step 4: Table Configuration
Is that it?
Two Switches We Studied

- **RMT**
  - 32 Stages
  - (SIGCOMM 2013)

- **FlexPipe**
  - 5 Stages
  - (Intel FM6000)
Additional switch features

- L2
- v4
- v6
- ACL

Table shaping in RMT

Table sharing in FlexPipe
The Compiler Problem

Map match action tables in a TDG to a switch pipeline while respecting dependency and resource constraints.
First approach: Greedy

- Prioritize one constraint
- Sort tables
- Map tables one at a time

Sort by # dependencies
First approach: Greedy

- Prioritize one constraint
- Sort tables
- Map tables one at a time
Too many constraints for Greedy

• Any greedy must sort tables based on a metric that is a *fixed* function of constraints.
• As the number of constraints gets larger, it’s harder for a fixed function to represent the interplay between all constraints.
• Can we do better than greedy?
Second approach:
Integer Linear Programming (ILP)

Find an optimal mapping.

**Pros:**
- Takes in all constraints
- Different objectives
- Solvers exist (CPLEX)

**Cons:**
- Blackbox solver
- Encoding is an art
- Slow
ILP Setup

\[ \text{min } \# \text{ stages} \]

subject to:

\[
\begin{align*}
\text{table sizes assigned} & \geq \text{table sizes specified} \\
\text{memories assigned} & \leq \text{memories in physical stage}
\end{align*}
\]

dependency constraints
Experiment Setup

• 4 datacenter use cases from Intel, Barefoot

• Differ in tables, table sizes, and dependencies
Example Use Case

A Typical TDG

Configuration for RMT
Metrics: Greedy vs ILP

1. Ability to fit program in chip

2. Optimality

3. Runtime
Setup: Greedy vs ILP

1. Ability to fit: FlexPipe
   - Variants of use cases in 5-stage pipeline.

2. Optimality: RMT
   - Minimum stage, pipeline latency, power

3. Runtime: both switches
Results: Greedy vs ILP

1. Can Greedy fit my program?
   – Yes, if resources aplenty (RMT, 32 stages)
   – No, if resources constrained (FlexPipe, 5 stages), Can’t fit 25% of programs.

2. How close to optimal is Greedy?
   – 30% more time for packet to get through RMT pipeline.

3. Hmm.. looks like I need ILP. How slow is it?
   – 100x slower than Greedy
   – Reasonable if programs don’t change often.
If we have time, we should run ILP.
Use ILP to suggest best Greedy for program type.

Critical constraints
• Dependency critical: 16 $\rightarrow$ 13 stages
• Additional resource constraints less important

Critical resources
• TCAM memories critical: 16 $\rightarrow$ 14 stages
  – Results for one of our datacenter L2/L3 use cases
Conclusion

• **Challenge**: Parallelism and constraints in reconfigurable chips makes compiling difficult.

• **TDG**: highlights parallelism in program.

• **ILP**: better if enough time, fitting is critical, or objectives are complicated.

• **Best Greedy**: ILP can choose via notion of *critical* constraints and *critical* resources.
Thank you!

Research funded by AT&T, Intel, Open Networking Research Center.
ILP Run time

• Number of constraints? Not obvious. E.g., RMT
  – Min. stage: few secs.
  – Min. power: few secs.
  – Min. pipeline latency 10x slower

• Number of variables? How fine-grained is the resource assignment? E.g., FlexPipe
  – One match entry at a time: many days..
  – 100-500 match entries at a time: < 1 hr