P4LLVM: An LLVM based P4 Compiler

Tharun Kumar Dangeti, Venkata Keerthy Soundararajan, Ramakrishna Upadrasta

Indian Institute of Technology Hyderabad

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

First P4 European Workshop - P4EU

September 24th, 2018

Outline

- P4LLVM - LLVM based P4 compiler
  - Better optimizations = improve the runtime performance of the network
- Frontend
  - P4-16 code → LLVM Intermediate Representation (IR)
- Backend
  - LLVM IR → Architecture code
- JSON Backend
  - BMv2 is a software switch for prototyping purpose
  - The input to BMv2 is a JSON file
- P4LLVM - Current support
  - P4-16 → LLVM IR → JSON (BMv2 target)

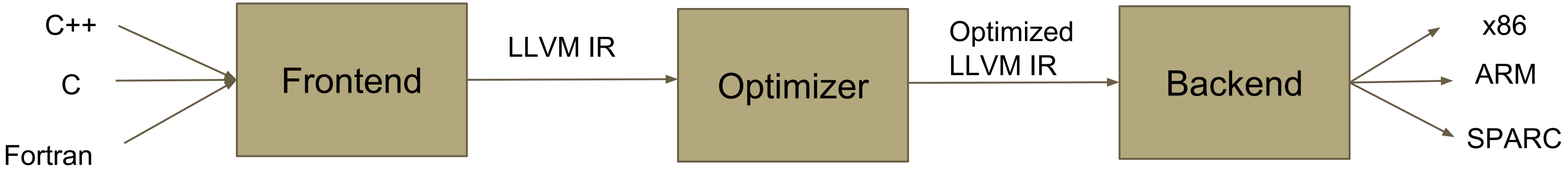
P4LLVM: Why?

- Input (switch) configuration affects the performance
    - Need for stronger compiler optimizations

- Compiler Optimizations
    - Theory Strong
    - Implementation
        - Evolves over time
        - Incremental development and difficult

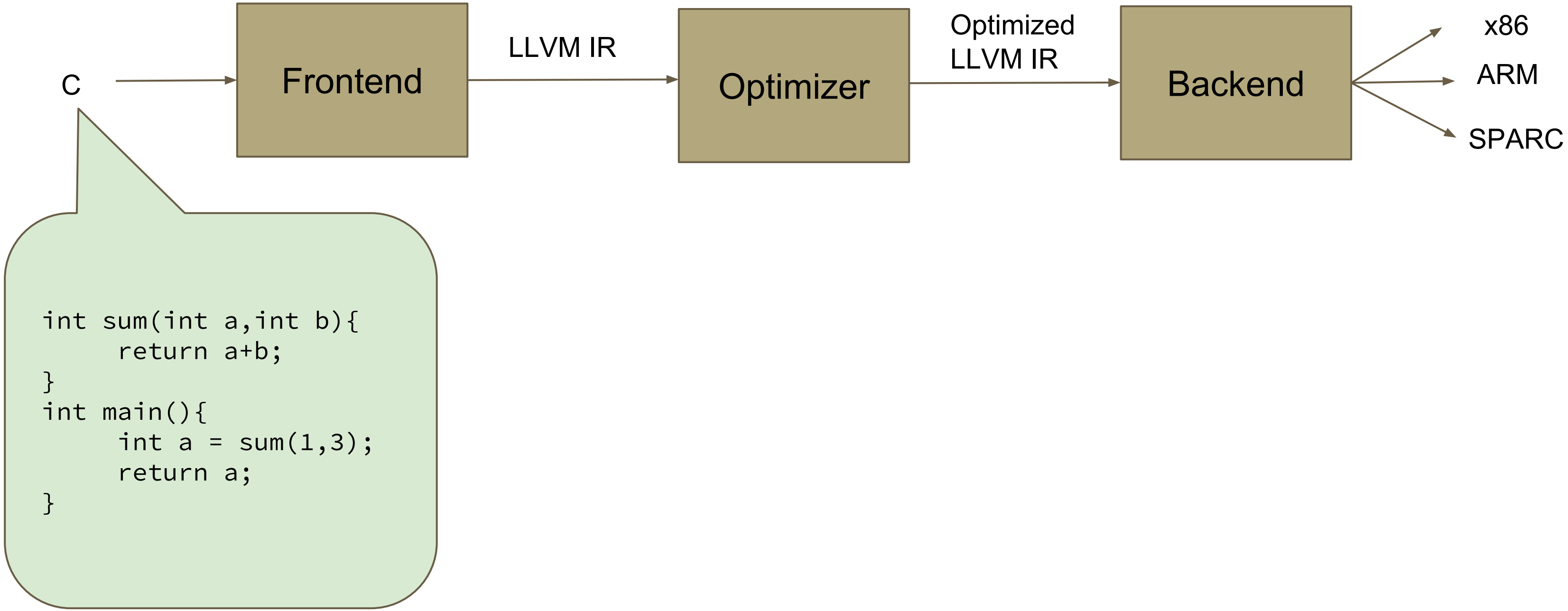
- Use existing & active compiler tool-chains
    - Mature enough
    - Relatively bug-free
    - Community support

What is LLVM?



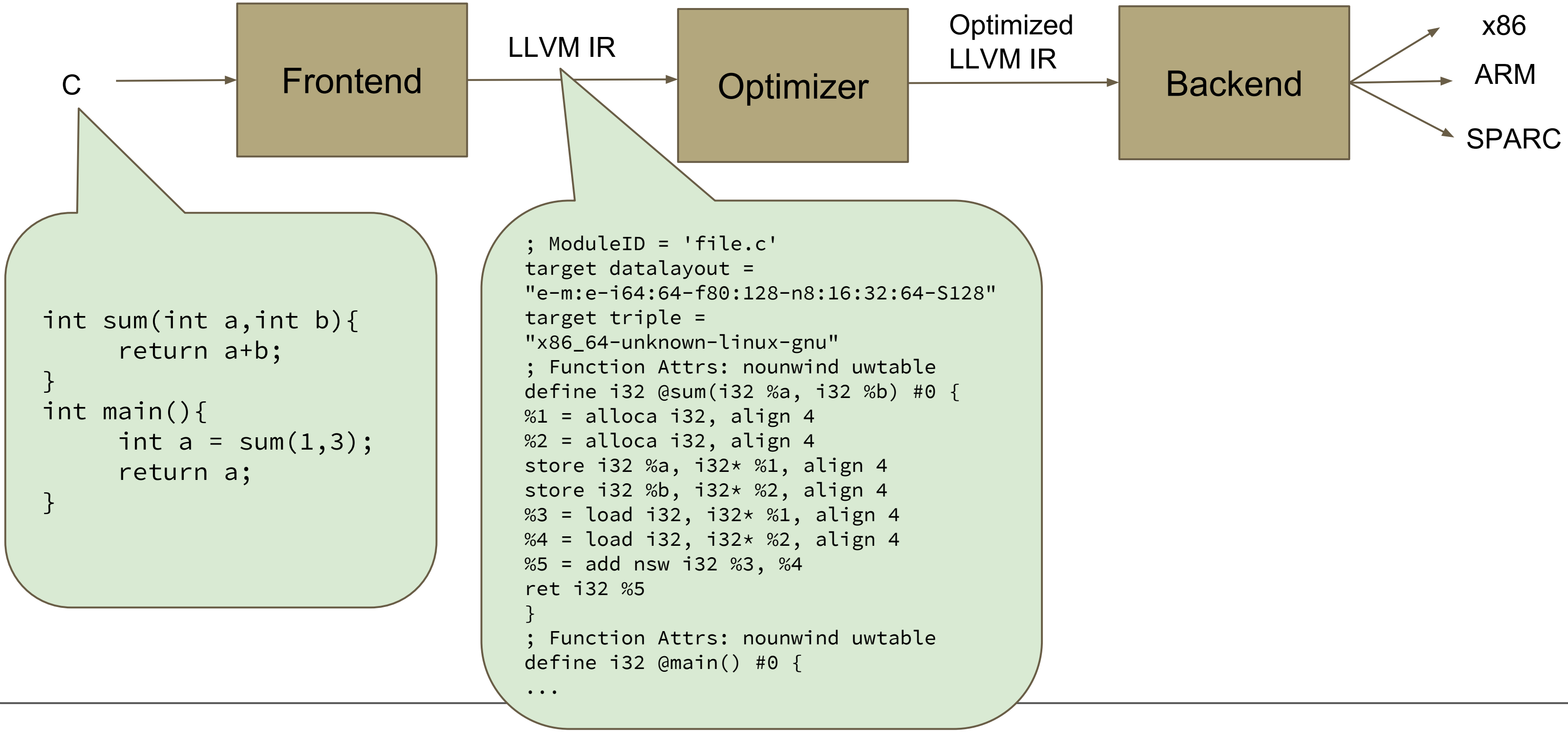
- Compiler infrastructure
- Written in C++
- Designed for Compile time, Link time and Run time optimizations.
- License: Allows LLVM to be used and sold in commercial tools.
- Flexible: Designed to be used like an API/library
  - GCC: more monolithic, cannot be used as a library

What is LLVM?



```
int sum(int a,int b){
     return a+b;
}
int main(){
     int a = sum(1,3);
     return a;
}
```

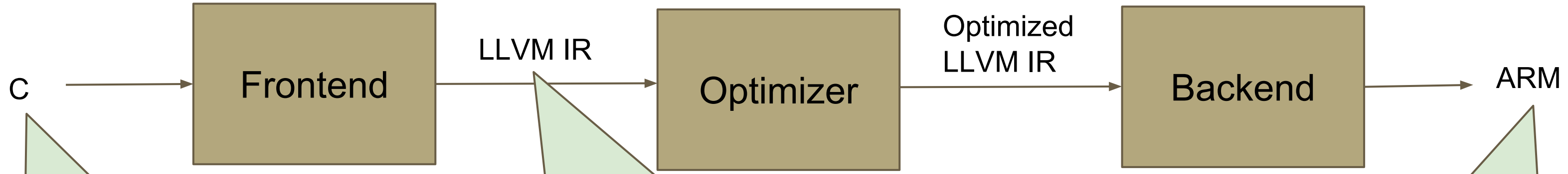
What is LLVM?



```
int sum(int a,int b){
      return a+b;
}
int main(){
      int a = sum(1,3);
      return a;
}
```

```
; ModuleID = 'file.c'
target datalayout =
"e-m:e-i64:64-f80:128-n8:16:32:64-S128"
target triple =
"x86_64-unknown-linux-gnu"
; Function Attrs: nounwind uwtable
define i32 @sum(i32 %a, i32 %b) #0 {
%1 = alloca i32, align 4
%2 = alloca i32, align 4
store i32 %a, i32* %1, align 4
store i32 %b, i32* %2, align 4
%3 = load i32, i32* %1, align 4
%4 = load i32, i32* %2, align 4
%5 = add nsw i32 %3, %4
ret i32 %5
}
; Function Attrs: nounwind uwtable
define i32 @main() #0 {
...
```

What is LLVM?

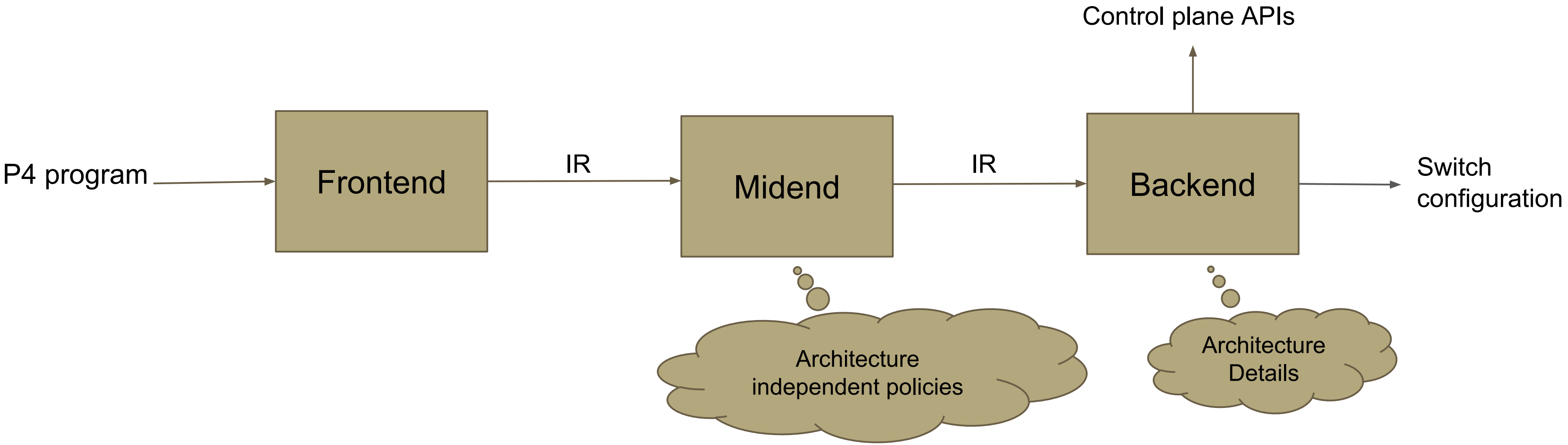


```
int sum(int a,int b){
     return a+b;
}
int main(){
     int a = sum(1,3);
     return a;
}
```

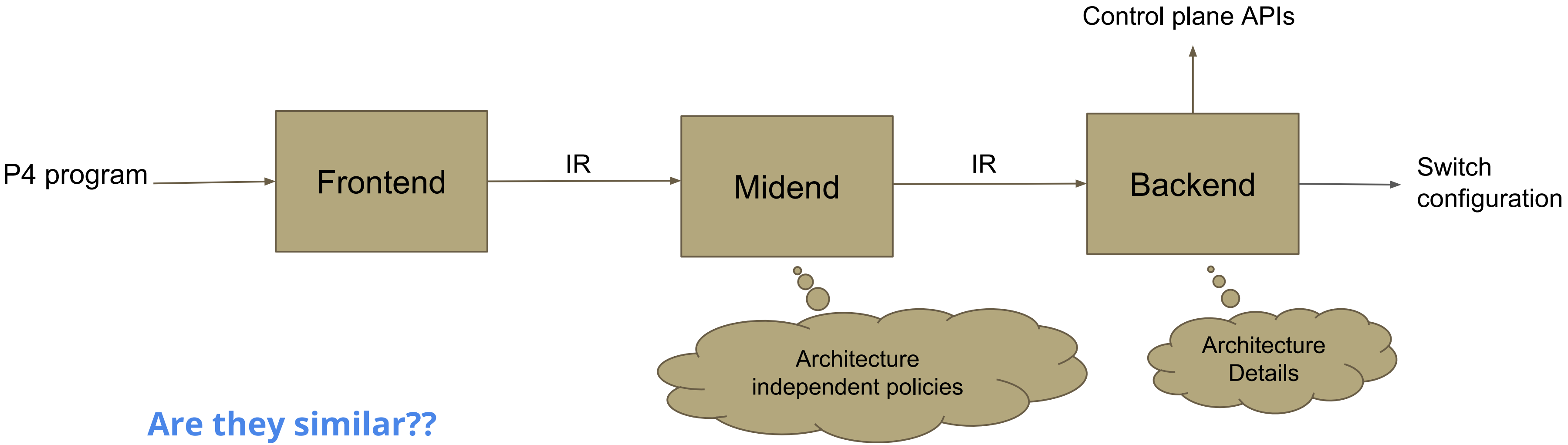
```
; ModuleID = 'file.c'
target datalayout =
"e-m:e-i64:64-f80:128-n8:16:32:64-S128"
target triple =
"x86_64-unknown-linux-gnu"
; Function Attrs: nounwind uwtable
define i32 @sum(i32 %a, i32 %b) #0 {
%1 = alloca i32, align 4
%2 = alloca i32, align 4
store i32 %a, i32* %1, align 4
store i32 %b, i32* %2, align 4
%3 = load i32, i32* %1, align 4
%4 = load i32, i32* %2, align 4
%5 = add nsw i32 %3, %4
ret i32 %5
}
; Function Attrs: nounwind uwtable
define i32 @main() #0 {
...
```

```
.text
.file "file.c"
.globl sum
.p2align
4, 0x90
.type sum,@function
sum:
# @sum
.cfi_startproc
# BB#0:
# %entry
pushq
%rbp
.Lcfi0:
.cfi_def_cfa_offset 16
.Lcfi1:
.cfi_offset %rbp, -16
movq
%rsp, %rbp
.Lcfi2:
.cfi_def_cfa_register
%rbp
movl
%edi, -4(%rbp)
movl
%esi, -8(%rbp)
...
```

A closer look at P4's compiler



A closer look at P4's compiler

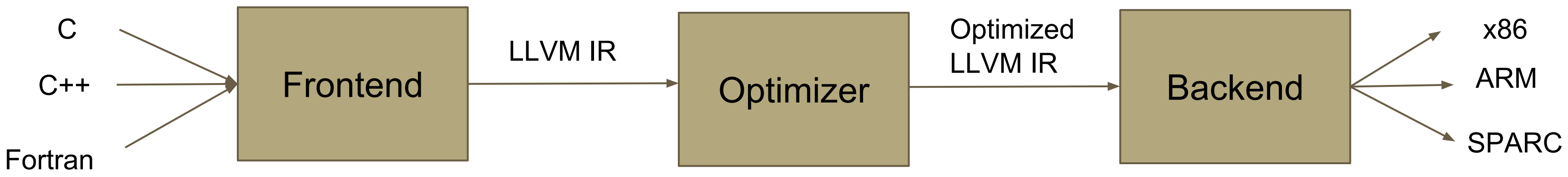


Are they similar??

YES!

Can P4 use LLVM?

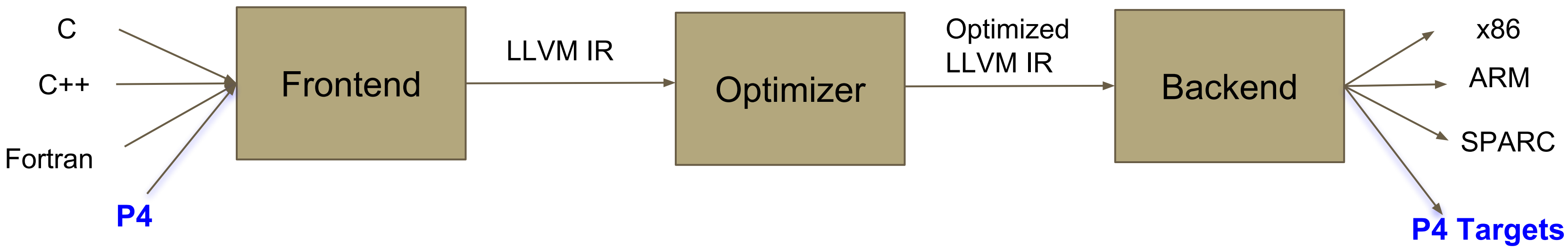
Recall



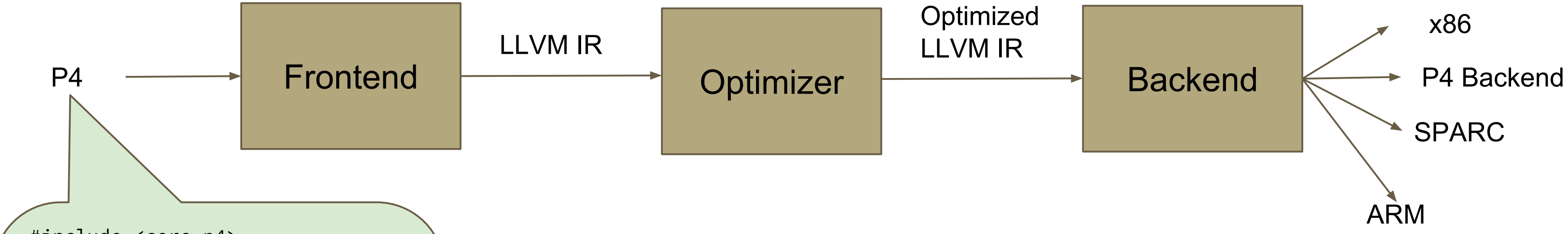
Can P4 use LLVM?

Recall

YES!

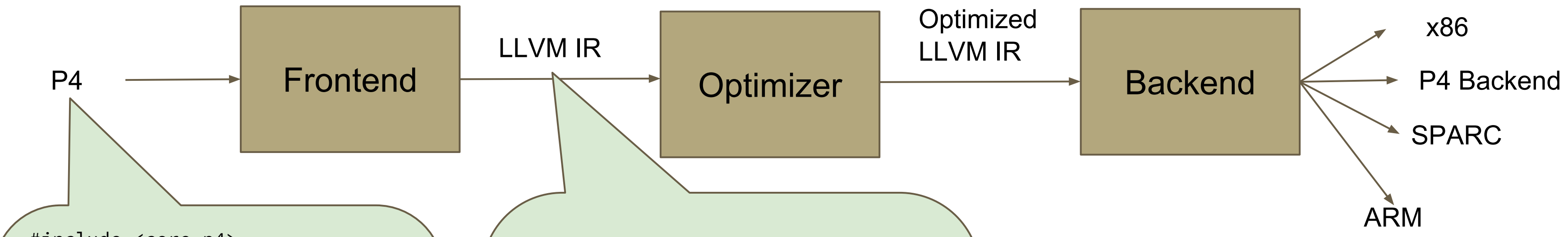


Can P4 use LLVM?



```
#include <core.p4>
#include <ebpf_model.p4>

struct Headers_t {}

parser prs(packet_in p, out
Headers_t headers) {
   state start {
       transition accept;
   }
}
```

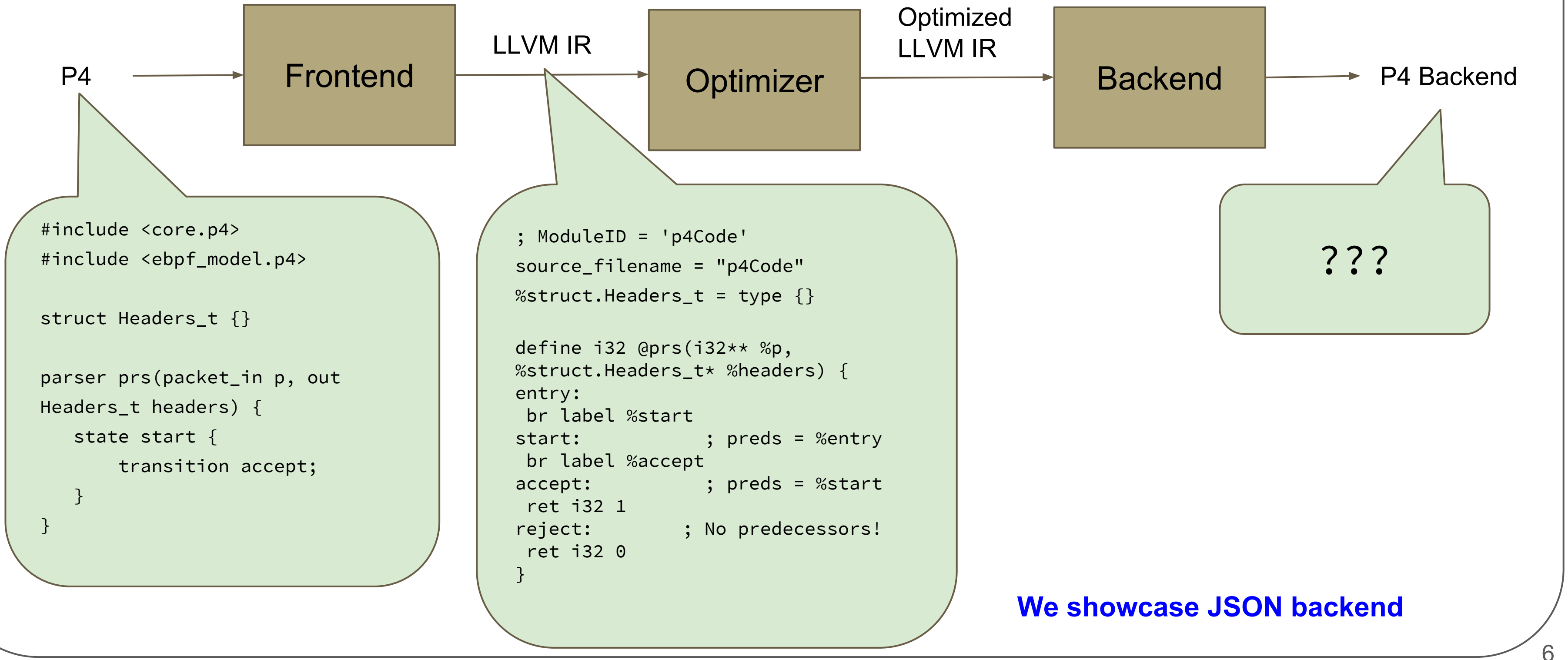
Can P4 use LLVM?



```
#include <core.p4>
#include <ebpf_model.p4>

struct Headers_t {}

parser prs(packet_in p, out
Headers_t headers) {
   state start {
       transition accept;
   }
}
```

```
; ModuleID = 'p4Code'
source_filename = "p4Code"
%struct.Headers_t = type {}

define i32 @prs(i32** %p,
%struct.Headers_t* %headers) {
entry:
 br label %start
start:           ; preds = %entry
 br label %accept
accept:          ; preds = %start
 ret i32 1
reject:        ; No predecessors!
 ret i32 0
}
```

Can P4 use LLVM?



P4 → Frontend → LLVM IR → Optimizer → Optimized LLVM IR → Backend → P4 Backend

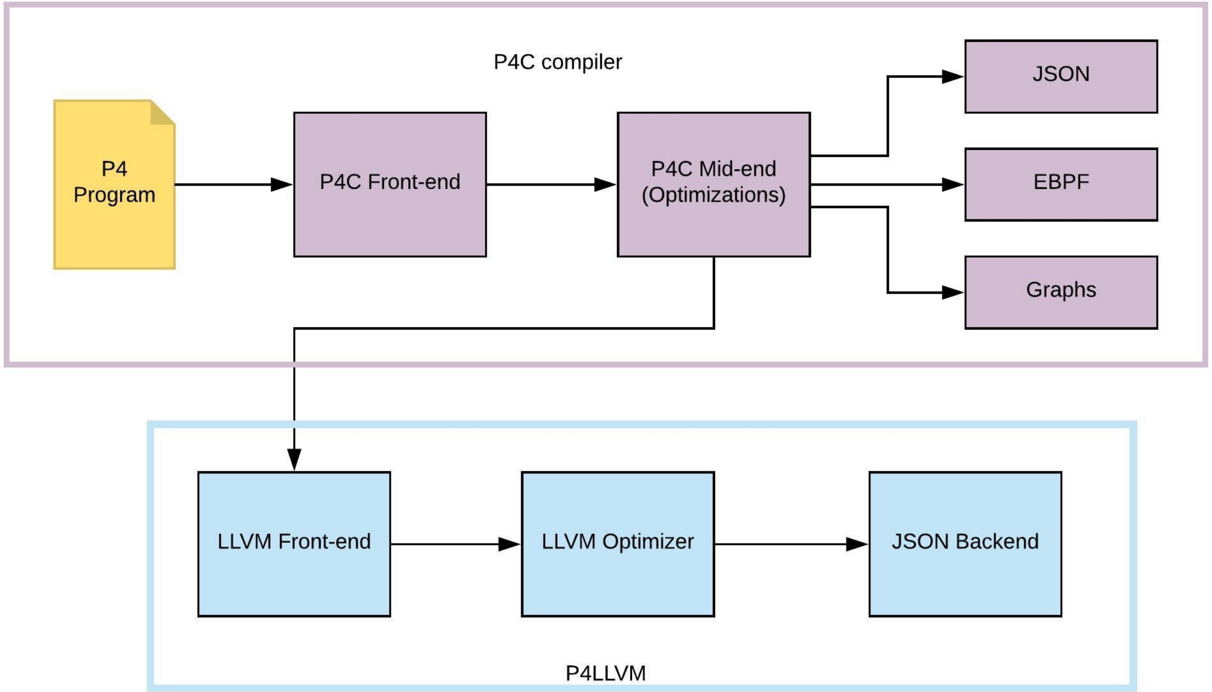
```
#include <core.p4>
#include <ebpf_model.p4>

struct Headers_t {}

parser prs(packet_in p, out
Headers_t headers) {
   state start {
       transition accept;
   }
}
```

```
; ModuleID = 'p4Code'
source_filename = "p4Code"
%struct.Headers_t = type {}

define i32 @prs(i32** %p,
%struct.Headers_t* %headers) {
entry:
 br label %start
start:           ; preds = %entry
 br label %accept
accept:          ; preds = %start
 ret i32 1
reject:        ; No predecessors!
 ret i32 0
}
```

???

We showcase JSON backend

Current architecture of P4LLVM



Representing P4 In LLVM-IR

P4 Construct	Equivalent LLVM IR Construct
Data types: Headers, Structs	Struct types
Data types: Header Union	Array of structs
Primitives: Int and Bit	Int and vector of 1 bit ints
Declarations	Alloca instructions
Assignments	Store instructions
Extern calls: Extract, Verify, SetValid/Invalid, IsValid, Apply	Function declaration and corresponding calls
Tables	Similar to Apply
Parser, Control, Action, Deparser	Functions
Direction: In	Passed by value
Direction: Out, InOut	Passed by reference

P4 IN LLVM-IR: Headers

```
struct Headers {
    Hdr h;
}

header hdr {
    int <32> a;
    int <32> b;
    bit <8> c;
}
```

P4 code

P4 IN LLVM-IR: Headers

```
struct Headers {
    Hdr h;
}

header hdr {
    int <32> a;
    int <32> b;
    bit <8> c;
}
```

P4 code

```
%struct.Headers = type { %struct.hdr }
%struct.hdr = type {i32, i32, <8 x i1>}
```

Equivalent LLVM IR

P4 IN LLVM-IR: Parser

```
parser ParserImpl(...) {

    state parse_ethernet {
        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType) {
            16w0x88f7: parse_ip;
            default: reject;
        }
    }
    state parse_ip {
        packet.extract(hdr.ip);
        transition select(hdr.ip.version) {
            16w4: accept;
            default: reject;
        }
    }
}
```

P4 IN LLVM-IR: Parser

```
parser ParserImpl(...) {

    state parse_ethernet {
        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType) {
            16w0x88f7: parse_ip;
            default: reject;
        }
    }
    state parse_ip {
        packet.extract(hdr.ip);
        transition select(hdr.ip.version) {
            16w4: accept;
            default: reject;
        }
    }
}
```

```
parse_ethernet:
  %0 = load %struct.headers, %struct.headers* %hdr
   %1 = getelementptr %struct.headers, %struct.headers*
           %hdr, i32 0, i32 0
   call void @extract(%struct.ethernet_t* %1)
   %2 = load %struct.headers, %struct.headers* %hdr
   %3 = getelementptr %struct.headers, %struct.headers*
           %hdr, i32 0, i32 0
   %4 = load %struct.ethernet_t, %struct.ethernet_t* %3
   %5 = load %struct.headers, %struct.headers* %hdr
   %6 = getelementptr %struct.headers, %struct.headers*
          %hdr, i32 0, i32 0
   %7 = getelementptr %struct.ethernet_t,
           %struct.ethernet_t* %6, i32 0, i32 2
   %8 = bitcast <16 x i1>* %7 to i16*
   %9 = load i16, i16* %8
   switch i16 %9, label %reject [
    i16 -30473, label %parse_ip
   ]
parse_ip:
   .......
```

P4 IN LLVM-IR: apply table

```
control ingress() {

    table forward_table {
        actions = {
            forward;
            _drop;
        }
        key = {
            hdr.ethernet.dstAddr: exact;
        }
        size = 4;
    }
    apply {
        forward_table.apply();
    }
}
```

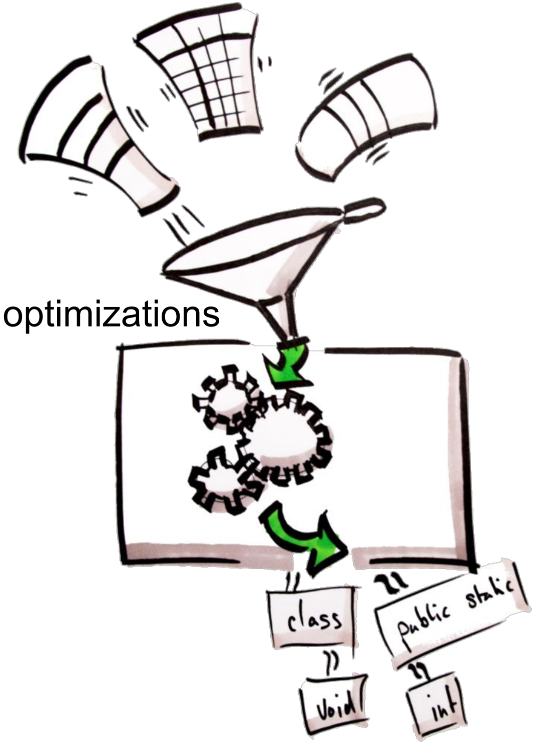
P4 IN LLVM-IR: apply table

```
control ingress() {

    table forward_table {
        actions = {
            forward;
            _drop;
        }
        key = {
            hdr.ethernet.dstAddr: exact;
        }
        size = 4;
    }
    apply {
        forward_table.apply();
    }
}
```

```
call void @apply_forward_table(
                “exact”,
                i48 %7,
                @forward,
                @_drop,
                @NoAction,
                i8* %8,
                i32 4,
                @NoAction
)
```

Code generation

- With LLVM the already available backends can be reused
  - Supports almost all CPUs and GPUs.
- LLVM provides sophisticated framework for addition of backends
- We have developed a JSON backend to show the effectiveness of optimizations
- Can target only *v1model* as of now
- Can be extended to any new switch models like PSA



Optimizations

- Numerous opportunities
  - Removing unused headers
  - Merging states with a single fan-out in parser
  - Eliminating dead/unreachable states
  - Removing unutilized extract calls

Optimizations

- Numerous opportunities
    - Removing unused headers
        - Dead code elimination - DCE
    - Merging states with a single fan-out in parser
        - Simplify CFG
    - Eliminating dead/unreachable states
        - Aggressive DCE
    - Removing unutilized extract calls

Optimizations

- Numerous opportunities
  - Removing unused headers
    - Dead code elimination - DCE
  - Merging states with a single fan-out in parser
    - Simplify CFG
  - Eliminating dead/unreachable states
    - Aggressive DCE
  - Removing unutilized extract calls
- SSA based optimizations
  - Simplifies and enables large set of optimizations
- Some trivial optimizations - available in P4C
  - Dead-state elimination
  - Constant propagation

Optimization - Opportunities

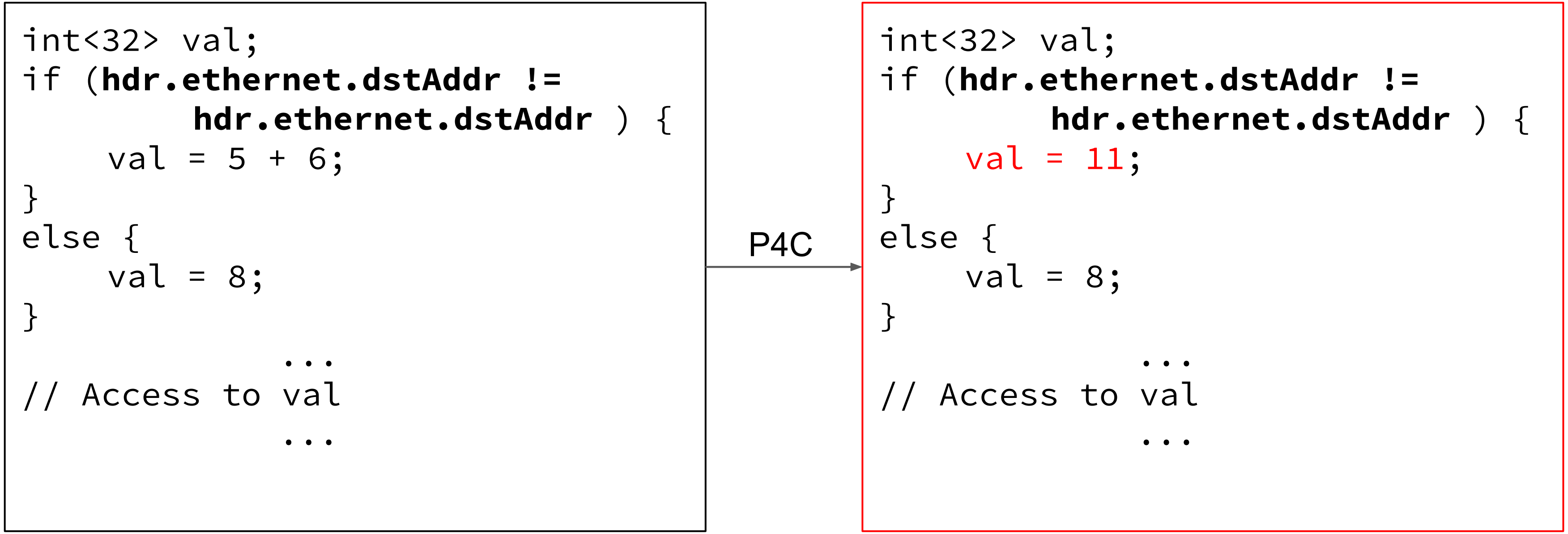
Case 1

```
int<32> val;
if (hdr.ethernet.dstAddr !=
        hdr.ethernet.dstAddr ) {
    val = 5 + 6;
}
else {
    val = 8;
}
             ...
// Access to val
             ...
```

P4 code

Optimization - Opportunities

Case 1



```
int<32> val;
if (hdr.ethernet.dstAddr !=
        hdr.ethernet.dstAddr ) {
    val = 5 + 6;
}
else {
    val = 8;
}
             ...
// Access to val
             ...
```

P4C

```
int<32> val;
if (hdr.ethernet.dstAddr !=
        hdr.ethernet.dstAddr ) {
    val = 11;
}
else {
    val = 8;
}
             ...
// Access to val
             ...
```

P4 code

Optimized by P4C

Optimization - Opportunities

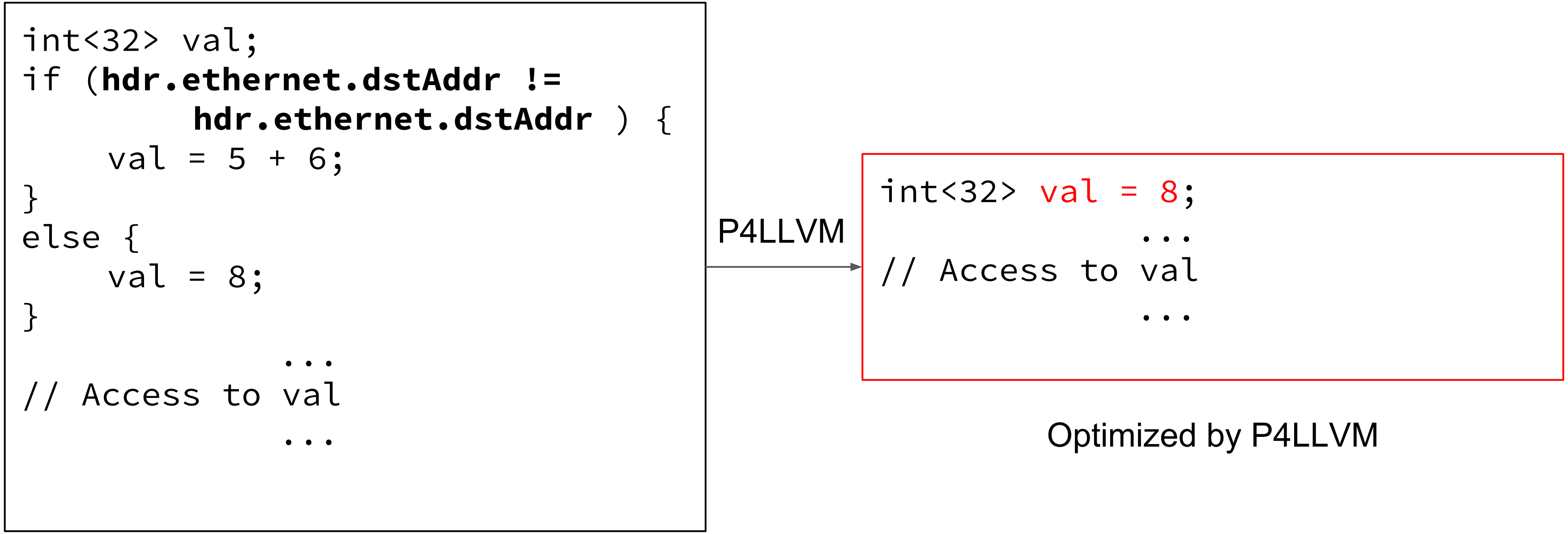
Case 1

```
int<32> val;
if (hdr.ethernet.dstAddr !=
        hdr.ethernet.dstAddr ) {
    val = 5 + 6;
}
else {
    val = 8;
}
             ...
// Access to val
             ...
```

P4 code

Optimization - Opportunities

Case 1



```
int<32> val;
if (hdr.ethernet.dstAddr !=
        hdr.ethernet.dstAddr ) {
    val = 5 + 6;
}
else {
    val = 8;
}
             ...
// Access to val
             ...
```

```
int<32> val = 8;
             ...
// Access to val
             ...
```

P4 code

Optimized by P4LLVM

Optimization - Opportunities

Case 2

```
struct hdr { … }
state parse_hdr {
             ...
    packet.extract(hdr);
             ...
    transition select (hdr.a) {
        default : accept;
    }
}
     <No further use of hdr>
```

Optimization - Opportunities

Case 2

```
struct hdr { … }
state parse_hdr {
            ...
    packet.extract(hdr);
            ...
    transition select (hdr.a) {
        default : accept;
    }
}
     <No further use of hdr>
```

`select` → more instructions than required

Can be removed

Optimization - Opportunities

Case 2

```
struct hdr { … }
state parse_hdr {
                ...
    packet.extract(hdr);
                ...
    transition select (hdr.a) {
        default : accept;
    }
}
     <No further use of hdr>
```

Unnecessary extraction

Can be removed

Optimization - Opportunities

Case 2

```
struct hdr { … }
state parse_hdr {
          ...
    packet.extract(hdr);
          ...
    transition select (hdr.a) {
        default : accept;
    }
}
     <No further use of hdr>
```

Unused header

Can be removed

Optimization - Opportunities

Case 2

```
struct hdr { … }
state parse_hdr {
           ...
     packet.extract(hdr);
           ...
     transition select (hdr.a) {
          default : accept;
     }
}
      <No further use of hdr>
```

Unused header

Can be removed

One optimization cascades to other optimizations

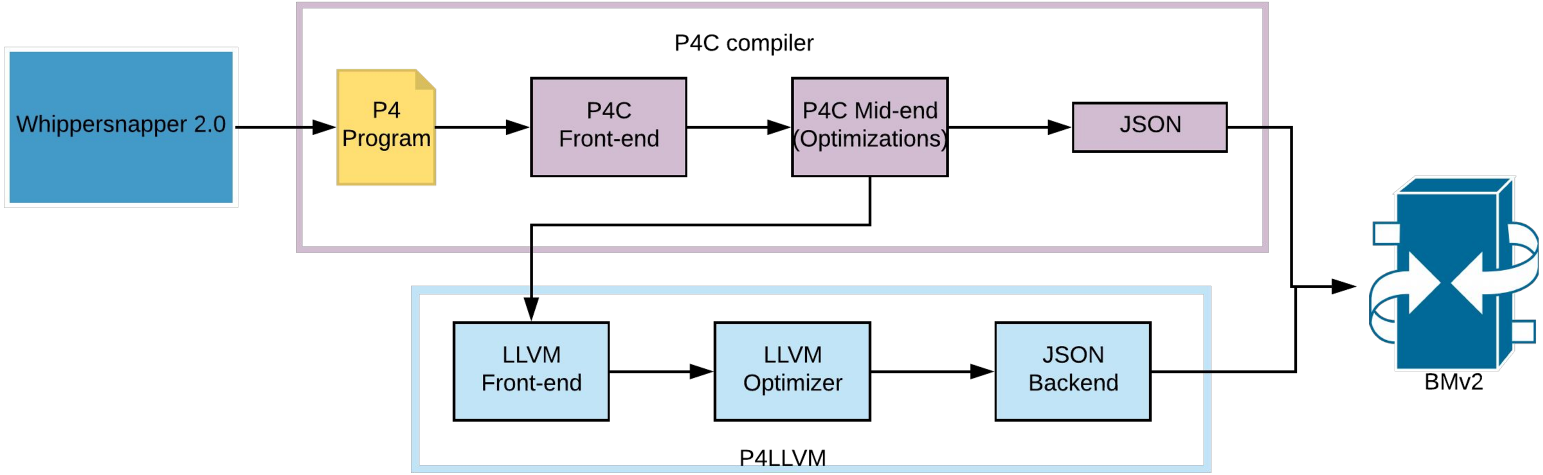
Experimentation

- Whippersnapper#
  - Benchmark suite to study the performance by P4 compilers
  - P4 compilers like P4C, P4FPGA, PISCES, Xilinx SDNet have been studied
- We extended to Whippersnapper 2.0*
  - To generate complex expressions, conditionals, etc.
  - So that optimizations can be studied
- We study performance using Whippersnapper 2.0 in comparison with P4C

*https://github.com/IITH-Compilers/Whippersnapper-2.0

#Dang, Huynh Tu, et al. "Whippersnapper: A p4 language benchmark suite." *Proceedings of the Symposium on SDN Research*. ACM, 2017.

Experimentation: Setup

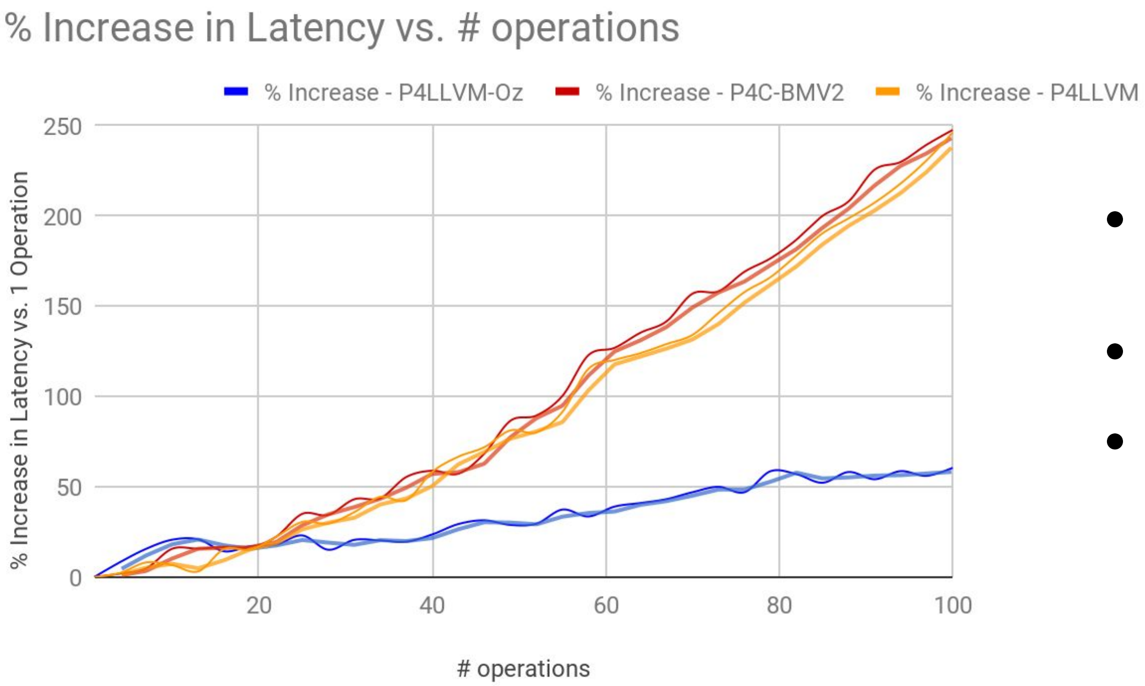


- LLVM IR optimized for size - Oz
- Latencies
  - Average of packet processing times of 10,000 packets

Optimization levels (LLVM 5.0.1)

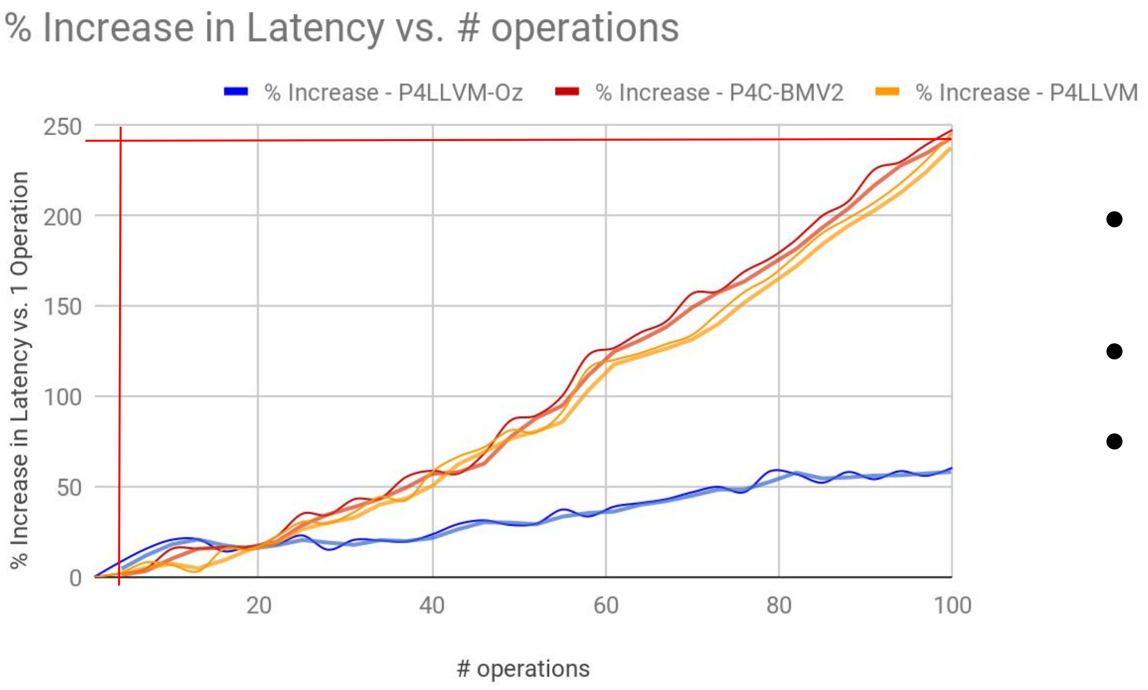
- **O1:** targetlibinfo tti tbaa scoped-noalias assumption-cache-tracker profile-summary-info forceattrs inferattrs ipsccp globalopt domtree mem2reg deadargelim basicaa aa instcombine simplifycfg basiccg globals-aa prune-eh always-inline functionattrs sroa memoryssa early-cse-memssa speculative-execution lazy-value-info jump-threading correlated-propagation libcalls-shrinkwrap loops branch-prob block-freq pgo-memop-opt tailcallelim reassociate loop-simplify lcssa-verification lcssa scalar-evolution loop-rotate licm loop-unswitch indvars loop-idiom loop-deletion loop-unroll memdep memcpyopt sccp demanded-bits bdce dse postdomtree adce barrier rpo-functionattrs float2int loop-accesses lazy-branch-prob lazy-block-freq opt-remark-emitter loop-distribute loop-vectorize loop-load-elim latesimplifycfg alignment-from-assumptions strip-dead-prototypes loop-sink instsimplify verify
- **O2:** O1 **+** constmerge **+** elim-avail-extern **+** globaldce **+** gvn **−** always-inline **+** inline **+** mldst-motion **+** slp-vectorizer
- **O3:** O2 **+** argpromotion
- **Os:** O2 **−** libcalls-shrinkwrap **−** pgo-memop-opt
- **Oz:** Os **−** slp-vectorizer

Results: Action Complexity



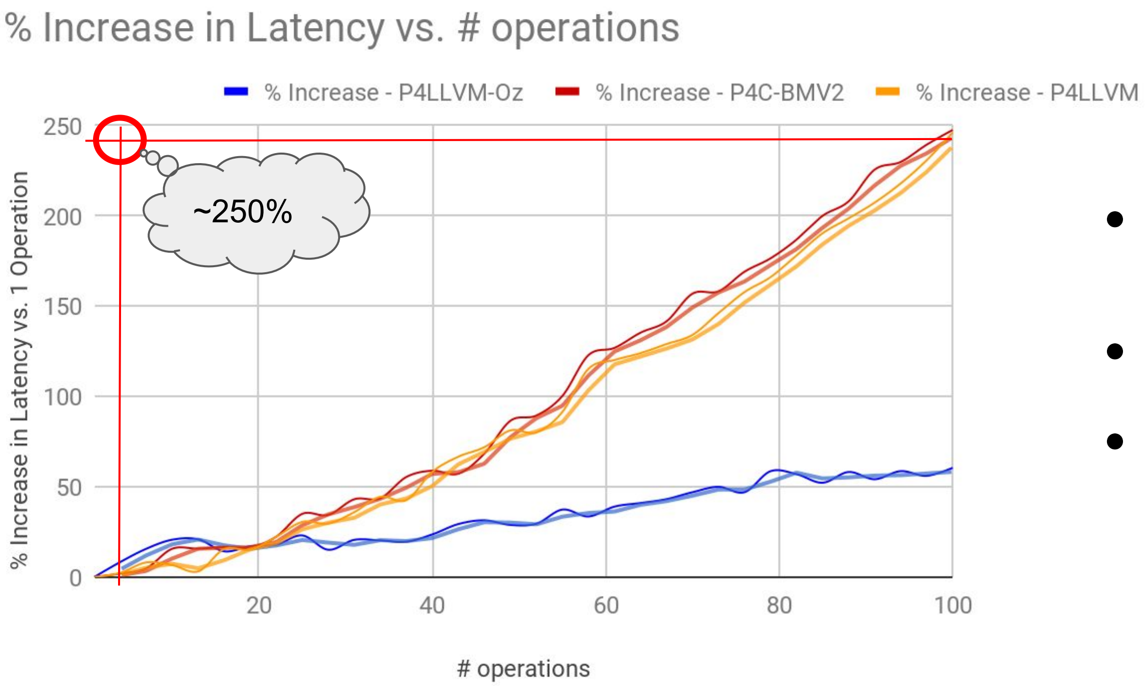
- P4C-BMV2: Existing P4C compiler
- P4LLVM: Without optimizations.
- P4LLVM-Oz: With ‘Oz’ optimization sequence of LLVM

Results: Action Complexity



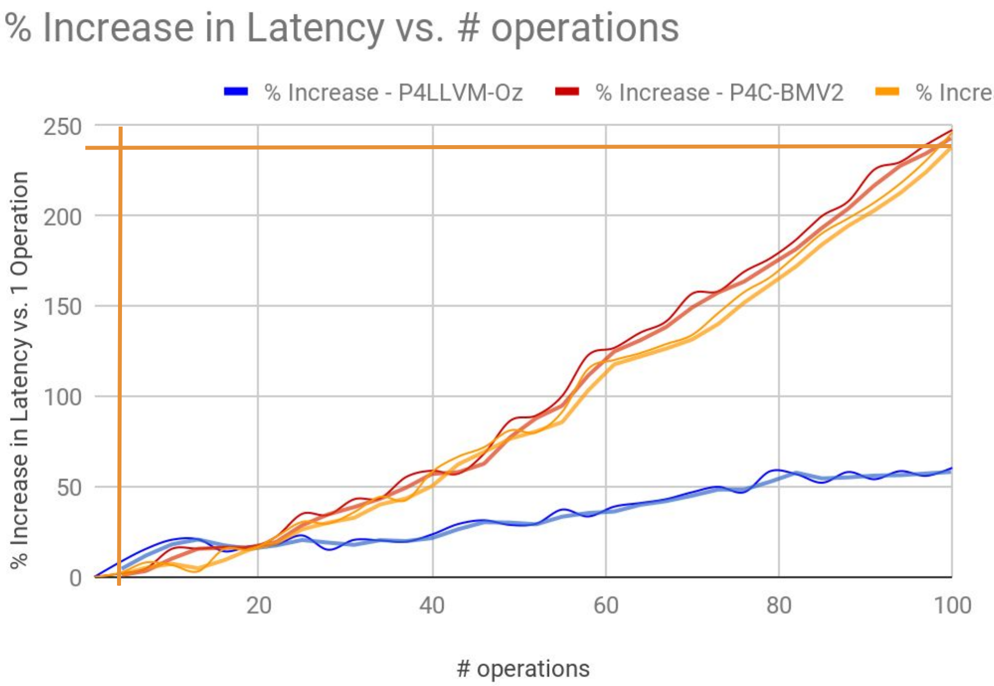
- P4C-BMV2: Existing P4C compiler
- P4LLVM: Without optimizations.
- P4LLVM-Oz: With 'Oz' optimization sequence of LLVM

Results: Action Complexity



- P4C-BMV2: Existing P4C compiler
- P4LLVM: Without optimizations.
- P4LLVM-Oz: With ‘Oz’ optimization sequence of LLVM

Results: Action Complexity

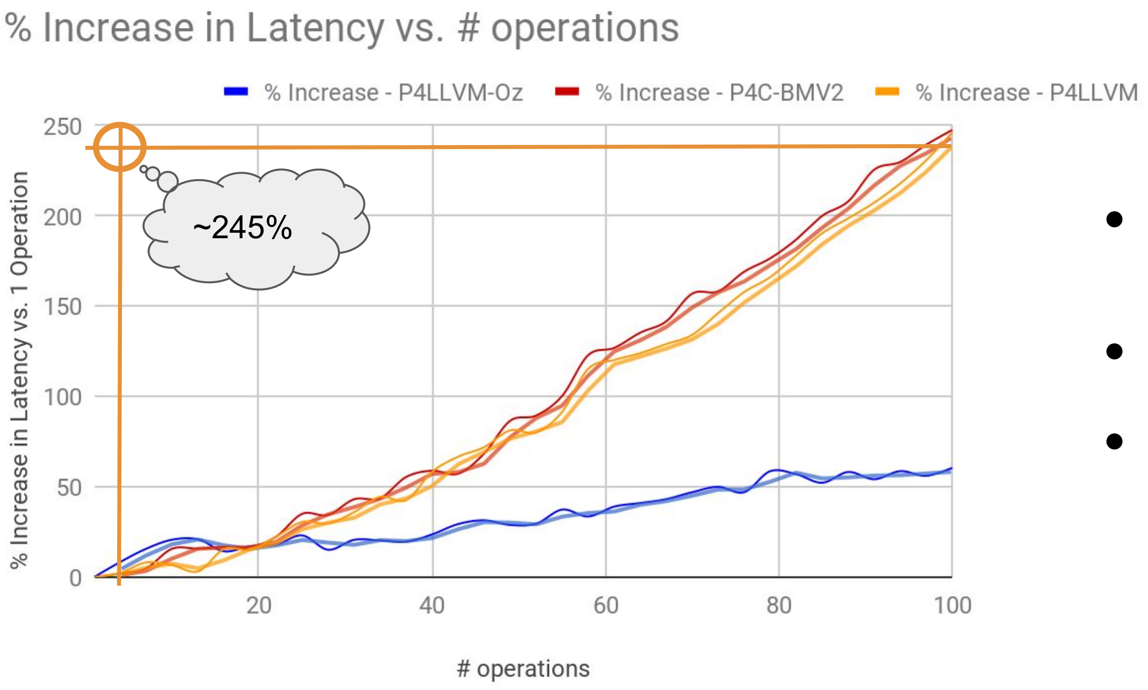


- P4C-BMV2: Existing P4C compiler

- P4LLVM: Without optimizations.

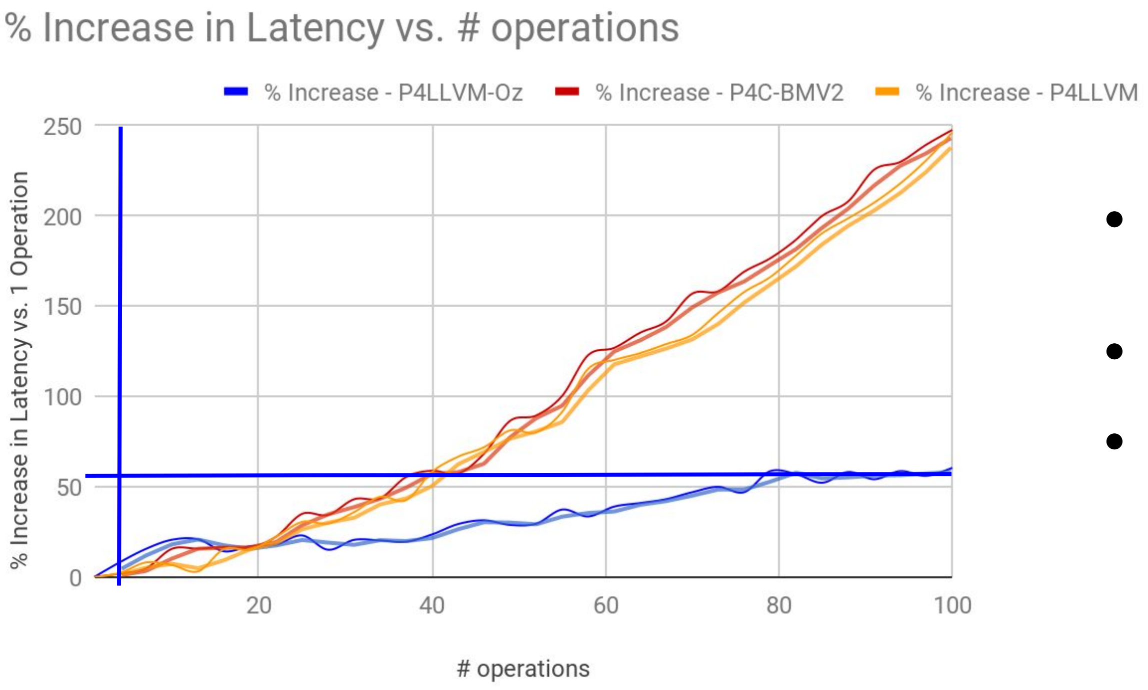
- P4LLVM-Oz: With ‘Oz’ optimization sequence of LLVM

Results: Action Complexity



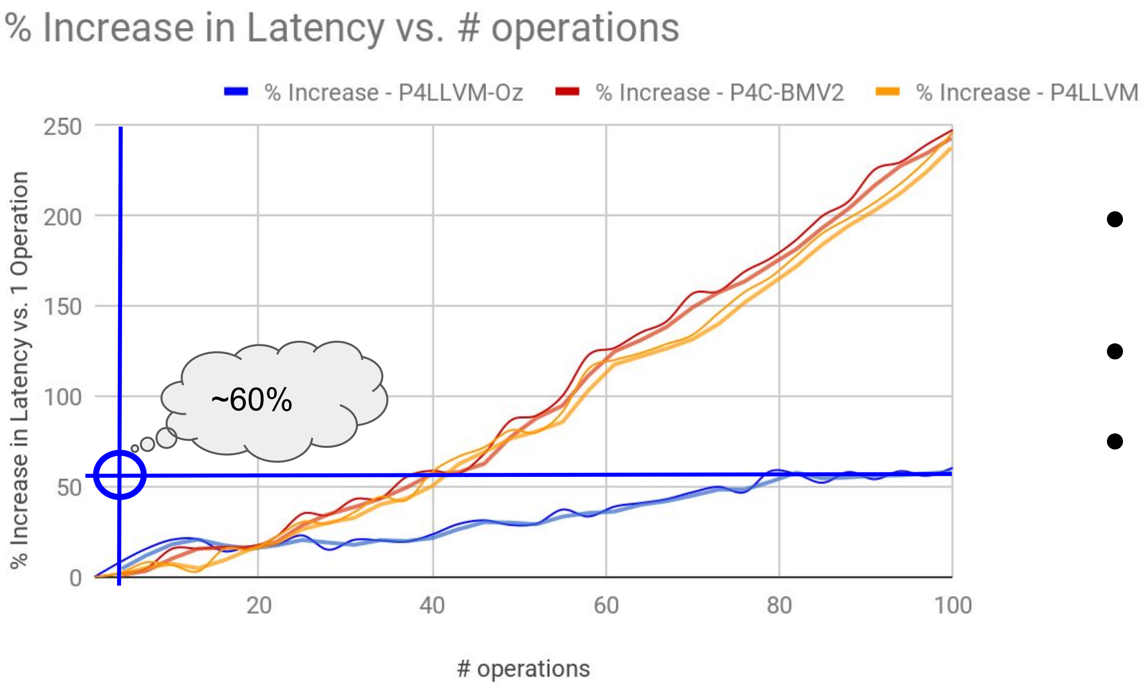
- P4C-BMV2: Existing P4C compiler
- P4LLVM: Without optimizations.
- P4LLVM-Oz: With ‘Oz’ optimization sequence of LLVM

Results: Action Complexity



- P4C-BMV2: Existing P4C compiler
- P4LLVM: Without optimizations.
- P4LLVM-Oz: With ‘Oz’ optimization sequence of LLVM

Results: Action Complexity

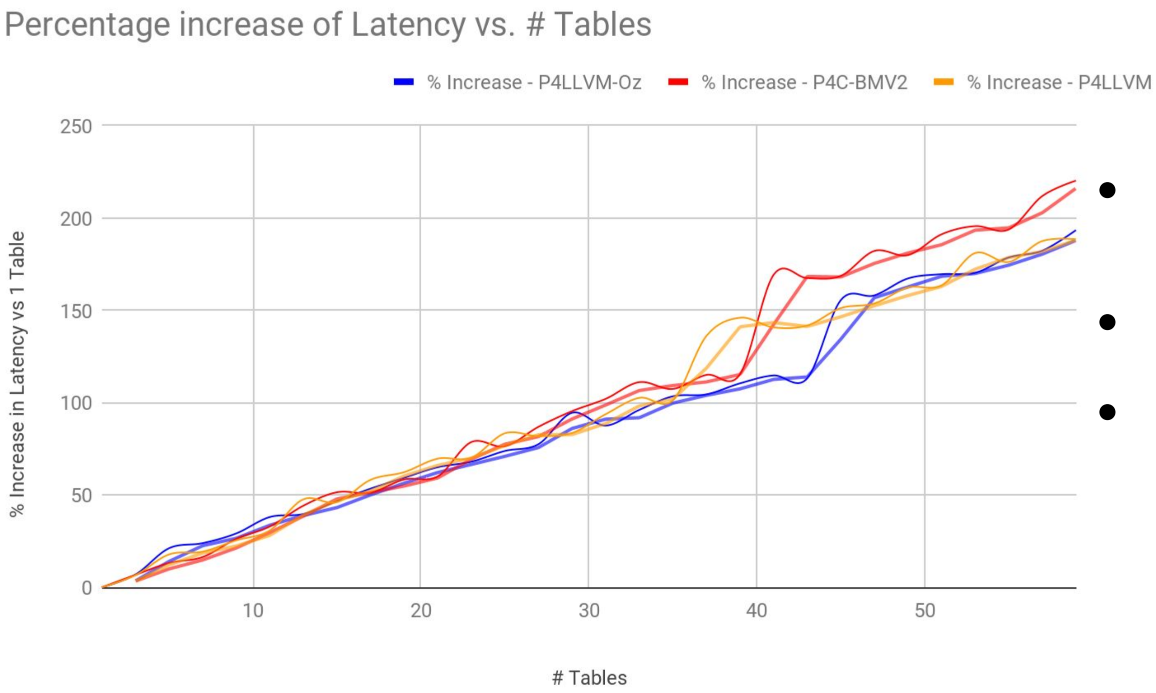


- P4C-BMV2: Existing P4C compiler

- P4LLVM: Without optimizations.

- P4LLVM-Oz: With ‘Oz’ optimization sequence of LLVM

Results: Table Depth



- P4C-BMV2: Existing P4C compiler
- P4LLVM: Without optimizations.
- P4LLVM-Oz: With ‘Oz’ optimization sequence of LLVM

Future Perspectives

- Tailoring a suitable pass sequence for P4
- Connecting with *P4runtime*
- Extending backend support for P4LLVM

Summary

- LLVM can fit with scope and spirit of P4 ⇒ P4LLVM

- Optimizations by P4LLVM show performance improvement
  - When compared to the established P4 compiler

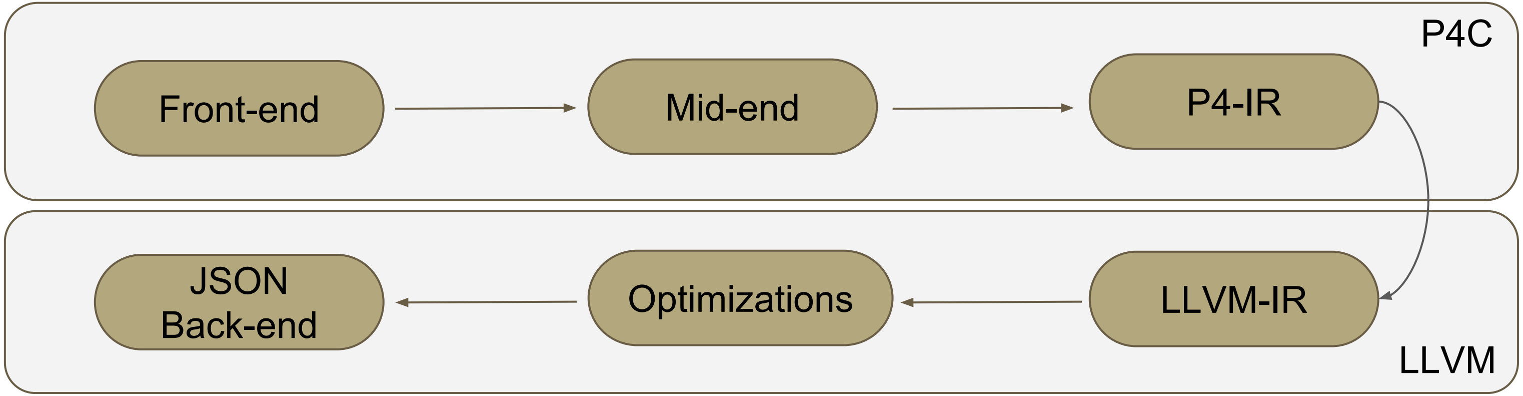
- *Target-independen*t optimizations with generic Oz sequence
  - More improvement → tailor *Target-specific* sequence

- Source code: https://github.com/IITH-Compilers/P4LLVM

Acknowledgements: Cisco India Team - *Raju Datla, Suresh Goduguluru, Sameek Banerjee*

Extra slides

How?



- Front-end module in P4C[*] to check syntactic and semantic correctness.
- We reuse P4 specific passes in the mid-end of P4C
  - *like simplifyKey*, *simplifySelectList*, *removeParameters*, etc.
  - these can be conveniently written within the LLVM framework.

*https://github.com/p4lang/p4c/tree/master/docs

Optimization levels (LLVM 5.0.1)

- **O2:** targetlibinfo tti tbaa scoped-noalias assumption-cache-tracker profile-summary-info forceattrs inferattrs ipsccp globalopt domtree mem2reg deadargelim basicaa aa instcombine simplifycfg basiccg globals-aa prune-eh inline functionattrs sroa memoryssa early-cse-memssa speculative-execution lazy-value-info jump-threading correlated-propagation **libcalls-shrinkwrap** loops branch-prob block-freq **pgo-memop-opt** tailcallelim reassociate loop-simplify lcssa-verification lcssa scalar-evolution loop-rotate licm loop-unswitch indvars loop-idiom loop-deletion loop-unroll mldst-motion memdep lazy-branch-prob lazy-block-freq opt-remark-emitter gvn memcpyopt sccp demanded-bits bdce dse postdomtree adce barrier elim-avail-extern rpo-functionattrs float2int loop-accesses loop-distribute loop-vectorize loop-load-elim **slp-vectorizer** latesimplifycfg alignment-from-assumptions strip-dead-prototypes globaldce constmerge loop-sink instsimplify verify

Optimization levels (LLVM 5.0.1)

- **Oz:** targetlibinfo tti tbaa scoped-noalias assumption-cache-tracker profile-summary-info forceattrs inferattrs ipsccp globalopt domtree mem2reg deadargelim basicaa aa instcombine simplifycfg basiccg globals-aa prune-eh inline functionattrs sroa memoryssa early-cse-memssa speculative-execution lazy-value-info jump-threading correlated-propagation loops branch-prob block-freq tailcallelim reassociate loop-simplify lcssa-verification lcssa scalar-evolution loop-rotate licm loop-unswitch indvars loop-idiom loop-deletion loop-unroll mldst-motion memdep lazy-branch-prob lazy-block-freq opt-remark-emitter gvn memcpyopt sccp demanded-bits bdce dse postdomtree adce barrier elim-avail-extern rpo-functionattrs float2int loop-accesses loop-distribute loop-vectorize loop-load-elim latesimplifycfg alignment-from-assumptions strip-dead-prototypes globaldce constmerge loop-sink instsimplify verify

Optimization levels (LLVM 5.0.1)

- **Oz:** targetlibinfo tti tbaa scoped-noalias assumption-cache-tracker profile-summary-info forceattrs inferattrs ipsccp globalopt domtree mem2reg deadargelim basicaa aa instcombine simplifycfg basiccg globals-aa prune-eh inline functionattrs sroa memoryssa early-cse-memssa speculative-execution lazy-value-info jump-threading correlated-propagation loops branch-prob block-freq tailcallelim reassociate loop-simplify lcssa-verification lcssa scalar-evolution loop-rotate licm loop-unswitch indvars loop-idiom loop-deletion loop-unroll mldst-motion memdep lazy-branch-prob lazy-block-freq opt-remark-emitter gvn memcpyopt sccp demanded-bits bdce dse postdomtree adce barrier elim-avail-extern rpo-functionattrs float2int loop-accesses loop-distribute loop-vectorize loop-load-elim latesimplifycfg alignment-from-assumptions strip-dead-prototypes globaldce constmerge loop-sink instsimplify verify

Oz sequence ⇒ optimizations to reduce code size