byoc2020

June 29, 2023

[ ]:
```
import tvm
from tvm import relay
```

Here we demonstrate how BYOC annotates a Relay graph.

Let's first define a simple Relay graph with supported and unsupported operators. This function includes a loop (control flow) to represent 3 convolution layers, although it's a bit weird to apply the same weights and biases many times...

[ ]:
```
def get_demo_mod():
    # Loop
    iter1 = relay.var("iter1", shape=(), dtype="int32")
    cond = relay.less(iter1, relay.const(2, dtype="int32"))
    inc = iter1 + relay.const(1, dtype="int32")
    loop_var = relay.var("while_loop")

    # Loop body
    d1 = relay.var("d1", shape=(1, 32, 56, 56), dtype="float32")
    w1 = relay.var("w1", shape=(32, 32, 3, 3), dtype="float32")
    b1 = relay.var("b1", shape=(32,), dtype="float32")
    conv = relay.nn.conv2d(d1, w1, strides=(1, 1), padding=(1, 1))
    bias = relay.nn.bias_add(conv, b1)
    relu = relay.nn.relu(bias)
    loop_cond_out = loop_var(inc, relu, w1, b1)

    conv = relay.nn.conv2d(d1, w1, strides=(1, 1), padding=(1, 1))
    bias = relay.nn.bias_add(conv, b1)
    relu = relay.nn.relu(bias)
    loop_break_out = relay.reshape(relu, (1, 56, 56, 32))

    ife = relay.If(cond, loop_cond_out, loop_break_out)

    data = relay.var("data", shape=(1, 32, 56, 56), dtype="float32")
    weight = relay.var("weight", shape=(32, 32, 3, 3), dtype="float32")
    bias = relay.var("bias", shape=(32,), dtype="float32")
    loop_func = relay.Function([iter1, d1, w1, b1], ife)
```

```
    out = relay.Let(loop_var, loop_func, loop_var(relay.const(0,␣
↪dtype="int32"), data, weight, bias))
    func = relay.Function([data, weight, bias], out)
    mod = tvm.IRModule.from_expr(func)
    mod = relay.transform.InferType()(mod)
    return mod
```

[ ]:
```
mod = get_demo_mod()
print(mod["main"].astext(show_meta_data=False))
```

```
#[version = "0.0.5"]
fn (%data: Tensor[(1, 32, 56, 56), float32] /* ty=Tensor[(1, 32, 56, 56),
float32] */, %weight: Tensor[(32, 32, 3, 3), float32] /* ty=Tensor[(32, 32, 3,
3), float32] */, %bias: Tensor[(32), float32] /* ty=Tensor[(32), float32] */) ->
Tensor[(1, 56, 56, 32), float32] {
  let %while_loop: fn (int32, Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32,
3, 3), float32], Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] /*
ty=fn (int32, Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3), float32],
Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] */ = fn (%iter1:
int32 /* ty=int32 */, %d1: Tensor[(1, 32, 56, 56), float32] /* ty=Tensor[(1, 32,
56, 56), float32] */, %w1: Tensor[(32, 32, 3, 3), float32] /* ty=Tensor[(32, 32,
3, 3), float32] */, %b1: Tensor[(32), float32] /* ty=Tensor[(32), float32] */)
-> Tensor[(1, 56, 56, 32), float32] {
    %0 = less(%iter1, 2 /* ty=int32 */) /* ty=bool */;
    if (%0) {
      %1 = nn.conv2d(%d1, %w1, padding=[1, 1, 1, 1]) /* ty=Tensor[(1, 32, 56,
56), float32] */;
      %2 = nn.bias_add(%1, %b1) /* ty=Tensor[(1, 32, 56, 56), float32] */;
      %3 = add(%iter1, 1 /* ty=int32 */) /* ty=int32 */;
      %4 = nn.relu(%2) /* ty=Tensor[(1, 32, 56, 56), float32] */;
      %while_loop(%3, %4, %w1, %b1) /* ty=Tensor[(1, 56, 56, 32), float32] */
    } else {
      %5 = nn.conv2d(%d1, %w1, padding=[1, 1, 1, 1]) /* ty=Tensor[(1, 32, 56,
56), float32] */;
      %6 = nn.bias_add(%5, %b1) /* ty=Tensor[(1, 32, 56, 56), float32] */;
      %7 = nn.relu(%6) /* ty=Tensor[(1, 32, 56, 56), float32] */;
      reshape(%7, newshape=[1, 56, 56, 32]) /* ty=Tensor[(1, 56, 56, 32),
float32] */
    }
  } /* ty=fn (int32, Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3),
float32], Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] */;
  %while_loop(0 /* ty=int32 */, %data, %weight, %bias) /* ty=Tensor[(1, 56, 56,
32), float32] */
} /* ty=fn (Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3), float32],
Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] */
```

```
[11:02:24] /home/qzylalala/work_space/tvm/src/ir/transform.cc:379: InferType:
Executing module pass with opt level: 0
```

Then we define the annotation rules. As we have mentioned in the presentation, developers can specify both operator-based and pattern-based annotation rules. Here, we define the single operators `reshape` and `add` are supported. In addition, we also define two supported patterns (Conv2D - (Bias) - ReLU).

[ ]:
```
demo_target = "byoc-target"
```

Operator-based annotation rules

[ ]:
```
@tvm.ir.register_op_attr("reshape", "target.byoc-target")
def reshape(expr):
    return True

@tvm.ir.register_op_attr("add", "target.byoc-target")
def add(expr):
    return True
```

Pattern-based annotation rules

[ ]:
```
def make_pattern(with_bias=True):
    from tvm.relay.dataflow_pattern import is_op, wildcard
    data = wildcard()
    weight = wildcard()
    bias = wildcard()
    conv = is_op("nn.conv2d")(data, weight)
    if with_bias:
        conv_out = is_op("nn.bias_add")(conv, bias)
    else:
        conv_out = conv
    return is_op("nn.relu")(conv_out)

conv2d_bias_relu_pat = ("byoc-target.conv2d_relu_with_bias",␣
 ↪make_pattern(with_bias=True))
conv2d_relu_pat = ("byoc-target.conv2d_relu_wo_bias",␣
 ↪make_pattern(with_bias=False))
patterns = [conv2d_bias_relu_pat, conv2d_relu_pat]
```

Now let's perform pattern-based annotation:

[ ]:
```
mod2 = relay.transform.MergeComposite(patterns)(mod)
print(mod2["main"].astext(show_meta_data=False))
```

```
#[version = "0.0.5"]
fn (%data: Tensor[(1, 32, 56, 56), float32] /* ty=Tensor[(1, 32, 56, 56),
float32] */, %weight: Tensor[(32, 32, 3, 3), float32] /* ty=Tensor[(32, 32, 3,
3), float32] */, %bias: Tensor[(32), float32] /* ty=Tensor[(32), float32] */) ->
Tensor[(1, 56, 56, 32), float32] {
  let %while_loop: fn (int32, Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32,
3, 3), float32], Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] /*
```

```
ty=fn (int32, Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3), float32],
Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] */ = fn (%iter1:
int32 /* ty=int32 */, %d1: Tensor[(1, 32, 56, 56), float32] /* ty=Tensor[(1, 32,
56, 56), float32] */, %w1: Tensor[(32, 32, 3, 3), float32] /* ty=Tensor[(32, 32,
3, 3), float32] */, %b1: Tensor[(32), float32] /* ty=Tensor[(32), float32] */)
-> Tensor[(1, 56, 56, 32), float32] {
    %0 = less(%iter1, 2 /* ty=int32 */) /* ty=bool */;
    if (%0) {
      %3 = fn (%FunctionVar_1_0: Tensor[(1, 32, 56, 56), float32] /*
ty=Tensor[(1, 32, 56, 56), float32] */, %FunctionVar_1_1: Tensor[(32, 32, 3, 3),
float32] /* ty=Tensor[(32, 32, 3, 3), float32] */, %FunctionVar_1_2:
Tensor[(32), float32] /* ty=Tensor[(32), float32] */,
PartitionedFromPattern="nn.conv2d_nn.bias_add_nn.relu_", Composite="byoc-
target.conv2d_relu_with_bias") -> Tensor[(1, 32, 56, 56), float32] {
        %1 = nn.conv2d(%FunctionVar_1_0, %FunctionVar_1_1, padding=[1, 1, 1, 1])
/* ty=Tensor[(1, 32, 56, 56), float32] */;
        %2 = nn.bias_add(%1, %FunctionVar_1_2) /* ty=Tensor[(1, 32, 56, 56),
float32] */;
        nn.relu(%2) /* ty=Tensor[(1, 32, 56, 56), float32] */
      } /* ty=fn (Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3),
float32], Tensor[(32), float32]) -> Tensor[(1, 32, 56, 56), float32] */;
      %4 = add(%iter1, 1 /* ty=int32 */) /* ty=int32 */;
      %5 = %3(%d1, %w1, %b1) /* ty=Tensor[(1, 32, 56, 56), float32] */;
      %while_loop(%4, %5, %w1, %b1) /* ty=Tensor[(1, 56, 56, 32), float32] */
    } else {
      %8 = fn (%FunctionVar_0_0: Tensor[(1, 32, 56, 56), float32] /*
ty=Tensor[(1, 32, 56, 56), float32] */, %FunctionVar_0_1: Tensor[(32, 32, 3, 3),
float32] /* ty=Tensor[(32, 32, 3, 3), float32] */, %FunctionVar_0_2:
Tensor[(32), float32] /* ty=Tensor[(32), float32] */,
PartitionedFromPattern="nn.conv2d_nn.bias_add_nn.relu_", Composite="byoc-
target.conv2d_relu_with_bias") -> Tensor[(1, 32, 56, 56), float32] {
        %6 = nn.conv2d(%FunctionVar_0_0, %FunctionVar_0_1, padding=[1, 1, 1, 1])
/* ty=Tensor[(1, 32, 56, 56), float32] */;
        %7 = nn.bias_add(%6, %FunctionVar_0_2) /* ty=Tensor[(1, 32, 56, 56),
float32] */;
        nn.relu(%7) /* ty=Tensor[(1, 32, 56, 56), float32] */
      } /* ty=fn (Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3),
float32], Tensor[(32), float32]) -> Tensor[(1, 32, 56, 56), float32] */;
      %9 = %8(%d1, %w1, %b1) /* ty=Tensor[(1, 32, 56, 56), float32] */;
      reshape(%9, newshape=[1, 56, 56, 32]) /* ty=Tensor[(1, 56, 56, 32),
float32] */
    }
  } /* ty=fn (int32, Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3),
float32], Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] */;
  %while_loop(0 /* ty=int32 */, %data, %weight, %bias) /* ty=Tensor[(1, 56, 56,
32), float32] */
} /* ty=fn (Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3), float32],
Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] */
```

```
[11:02:32] /home/qzylalala/work_space/tvm/src/relay/ir/transform.cc:124:
MergeComposite: Executing function pass with opt level: 0
[11:02:32] /home/qzylalala/work_space/tvm/src/relay/ir/transform.cc:125:
MergeComposite: Input module:
def @main(%data: Tensor[(1, 32, 56, 56), float32] /* ty=Tensor[(1, 32, 56, 56),
float32] */, %weight: Tensor[(32, 32, 3, 3), float32] /* ty=Tensor[(32, 32, 3,
3), float32] */, %bias: Tensor[(32), float32] /* ty=Tensor[(32), float32] */) ->
Tensor[(1, 56, 56, 32), float32] {
  let %while_loop: fn (int32, Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32,
3, 3), float32], Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] /*
ty=fn (int32, Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3), float32],
Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] */ = fn (%iter1:
int32 /* ty=int32 */, %d1: Tensor[(1, 32, 56, 56), float32] /* ty=Tensor[(1, 32,
56, 56), float32] */, %w1: Tensor[(32, 32, 3, 3), float32] /* ty=Tensor[(32, 32,
3, 3), float32] */, %b1: Tensor[(32), float32] /* ty=Tensor[(32), float32] */)
-> Tensor[(1, 56, 56, 32), float32] {
    %0 = less(%iter1, 2 /* ty=int32 */) /* ty=bool */;
    if (%0) {
      %1 = nn.conv2d(%d1, %w1, padding=[1, 1, 1, 1]) /* ty=Tensor[(1, 32, 56,
56), float32] */;
      %2 = nn.bias_add(%1, %b1) /* ty=Tensor[(1, 32, 56, 56), float32] */;
      %3 = add(%iter1, 1 /* ty=int32 */) /* ty=int32 */;
      %4 = nn.relu(%2) /* ty=Tensor[(1, 32, 56, 56), float32] */;
      %while_loop(%3, %4, %w1, %b1) /* ty=Tensor[(1, 56, 56, 32), float32] */
    } else {
      %5 = nn.conv2d(%d1, %w1, padding=[1, 1, 1, 1]) /* ty=Tensor[(1, 32, 56,
56), float32] */;
      %6 = nn.bias_add(%5, %b1) /* ty=Tensor[(1, 32, 56, 56), float32] */;
      %7 = nn.relu(%6) /* ty=Tensor[(1, 32, 56, 56), float32] */;
      reshape(%7, newshape=[1, 56, 56, 32]) /* ty=Tensor[(1, 56, 56, 32),
float32] */
    }
  } /* ty=fn (int32, Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3),
float32], Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] */;
  %while_loop(0 /* ty=int32 */, %data, %weight, %bias) /* ty=Tensor[(1, 56, 56,
32), float32] */
}

[11:02:32] /home/qzylalala/work_space/tvm/src/ir/transform.cc:379:
MergeComposite: InferType: Executing module pass with opt level: 0
[11:02:32] /home/qzylalala/work_space/tvm/src/ir/transform.cc:379:
MergeComposite: InferType: Executing module pass with opt level: 0
[11:02:32] /home/qzylalala/work_space/tvm/src/relay/ir/transform.cc:148:
MergeComposite: Output module:
def @main(%data: Tensor[(1, 32, 56, 56), float32] /* ty=Tensor[(1, 32, 56, 56),
float32] */, %weight: Tensor[(32, 32, 3, 3), float32] /* ty=Tensor[(32, 32, 3,
3), float32] */, %bias: Tensor[(32), float32] /* ty=Tensor[(32), float32] */) ->
Tensor[(1, 56, 56, 32), float32] {
```

```
  let %while_loop: fn (int32, Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32,
3, 3), float32], Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] /*
ty=fn (int32, Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3), float32],
Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] */ = fn (%iter1:
int32 /* ty=int32 */, %d1: Tensor[(1, 32, 56, 56), float32] /* ty=Tensor[(1, 32,
56, 56), float32] */, %w1: Tensor[(32, 32, 3, 3), float32] /* ty=Tensor[(32, 32,
3, 3), float32] */, %b1: Tensor[(32), float32] /* ty=Tensor[(32), float32] */)
-> Tensor[(1, 56, 56, 32), float32] {
    %0 = less(%iter1, 2 /* ty=int32 */) /* ty=bool */;
    if (%0) {
      %3 = fn (%FunctionVar_1_0: Tensor[(1, 32, 56, 56), float32] /*
ty=Tensor[(1, 32, 56, 56), float32] */, %FunctionVar_1_1: Tensor[(32, 32, 3, 3),
float32] /* ty=Tensor[(32, 32, 3, 3), float32] */, %FunctionVar_1_2:
Tensor[(32), float32] /* ty=Tensor[(32), float32] */,
PartitionedFromPattern="nn.conv2d_nn.bias_add_nn.relu_", Composite="byoc-
target.conv2d_relu_with_bias") -> Tensor[(1, 32, 56, 56), float32] {
        %1 = nn.conv2d(%FunctionVar_1_0, %FunctionVar_1_1, padding=[1, 1, 1, 1])
/* ty=Tensor[(1, 32, 56, 56), float32] */;
        %2 = nn.bias_add(%1, %FunctionVar_1_2) /* ty=Tensor[(1, 32, 56, 56),
float32] */;
        nn.relu(%2) /* ty=Tensor[(1, 32, 56, 56), float32] */
      } /* ty=fn (Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3),
float32], Tensor[(32), float32]) -> Tensor[(1, 32, 56, 56), float32] */;
      %4 = add(%iter1, 1 /* ty=int32 */) /* ty=int32 */;
      %5 = %3(%d1, %w1, %b1) /* ty=Tensor[(1, 32, 56, 56), float32] */;
      %while_loop(%4, %5, %w1, %b1) /* ty=Tensor[(1, 56, 56, 32), float32] */
    } else {
      %8 = fn (%FunctionVar_0_0: Tensor[(1, 32, 56, 56), float32] /*
ty=Tensor[(1, 32, 56, 56), float32] */, %FunctionVar_0_1: Tensor[(32, 32, 3, 3),
float32] /* ty=Tensor[(32, 32, 3, 3), float32] */, %FunctionVar_0_2:
Tensor[(32), float32] /* ty=Tensor[(32), float32] */,
PartitionedFromPattern="nn.conv2d_nn.bias_add_nn.relu_", Composite="byoc-
target.conv2d_relu_with_bias") -> Tensor[(1, 32, 56, 56), float32] {
        %6 = nn.conv2d(%FunctionVar_0_0, %FunctionVar_0_1, padding=[1, 1, 1, 1])
/* ty=Tensor[(1, 32, 56, 56), float32] */;
        %7 = nn.bias_add(%6, %FunctionVar_0_2) /* ty=Tensor[(1, 32, 56, 56),
float32] */;
        nn.relu(%7) /* ty=Tensor[(1, 32, 56, 56), float32] */
      } /* ty=fn (Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3),
float32], Tensor[(32), float32]) -> Tensor[(1, 32, 56, 56), float32] */;
      %9 = %8(%d1, %w1, %b1) /* ty=Tensor[(1, 32, 56, 56), float32] */;
      reshape(%9, newshape=[1, 56, 56, 32]) /* ty=Tensor[(1, 56, 56, 32),
float32] */
    }
  } /* ty=fn (int32, Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3),
float32], Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] */;
  %while_loop(0 /* ty=int32 */, %data, %weight, %bias) /* ty=Tensor[(1, 56, 56,
32), float32] */
```

```
}

[11:02:32] /home/qzylalala/work_space/tvm/src/ir/transform.cc:379:
MergeComposite: InferType: Executing module pass with opt level: 0
```

We can see that now all subgraphs that match the defined patterns are partitioned to ”composite functions”. In this example, we got two composite functions.

A composite function has two specialized attributes -- ”PartitionedFromPattern” and ”Composite”: * PartitionedFromPattern: Indicate the operators in the function body. * Composite: Indicate the pattern name we defined.

As you can imagine, this information could be useful for you to map a composite function to your accelerator in the codegen. Next, let’s continue to apply the operator-based annotation rules:

[ ]:
```
mod3 = relay.transform.AnnotateTarget("byoc-target")(mod2)
print(mod3["main"].astext(show_meta_data=False))
```

```
#[version = "0.0.5"]
fn (%data: Tensor[(1, 32, 56, 56), float32] /* ty=Tensor[(1, 32, 56, 56),
float32] */, %weight: Tensor[(32, 32, 3, 3), float32] /* ty=Tensor[(32, 32, 3,
3), float32] */, %bias: Tensor[(32), float32] /* ty=Tensor[(32), float32] */) ->
Tensor[(1, 56, 56, 32), float32] {
  let %while_loop: fn (int32, Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32,
3, 3), float32], Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] /*
ty=fn (int32, Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3), float32],
Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] */ = fn (%iter1:
int32 /* ty=int32 */, %d1: Tensor[(1, 32, 56, 56), float32] /* ty=Tensor[(1, 32,
56, 56), float32] */, %w1: Tensor[(32, 32, 3, 3), float32] /* ty=Tensor[(32, 32,
3, 3), float32] */, %b1: Tensor[(32), float32] /* ty=Tensor[(32), float32] */)
-> Tensor[(1, 56, 56, 32), float32] {
    %0 = annotation.compiler_begin(%iter1, compiler="default") /* ty=int32 */;
    %1 = annotation.compiler_begin(2 /* ty=int32 */, compiler="default") /*
ty=int32 */;
    %2 = less(%0, %1) /* ty=bool */;
    %3 = annotation.compiler_end(%2, compiler="default") /* ty=bool */;
    if (%3) {
      %4 = annotation.compiler_begin(%iter1, compiler="byoc-target") /* ty=int32
*/;
      %5 = annotation.compiler_begin(1 /* ty=int32 */, compiler="byoc-target")
/* ty=int32 */;
      %6 = add(%4, %5) /* ty=int32 */;
      %7 = annotation.compiler_end(%6, compiler="byoc-target") /* ty=int32 */;
      %10 = annotation.compiler_begin(%d1, compiler="byoc-target") /*
ty=Tensor[(1, 32, 56, 56), float32] */;
      %11 = annotation.compiler_begin(%w1, compiler="byoc-target") /*
ty=Tensor[(32, 32, 3, 3), float32] */;
      %12 = annotation.compiler_begin(%b1, compiler="byoc-target") /*
ty=Tensor[(32), float32] */;
```

```
      %13 = fn (%FunctionVar_1_0: Tensor[(1, 32, 56, 56), float32] /*
ty=Tensor[(1, 32, 56, 56), float32] */, %FunctionVar_1_1: Tensor[(32, 32, 3, 3),
float32] /* ty=Tensor[(32, 32, 3, 3), float32] */, %FunctionVar_1_2:
Tensor[(32), float32] /* ty=Tensor[(32), float32] */,
PartitionedFromPattern="nn.conv2d_nn.bias_add_nn.relu_", Composite="byoc-
target.conv2d_relu_with_bias") -> Tensor[(1, 32, 56, 56), float32] {
        %8 = nn.conv2d(%FunctionVar_1_0, %FunctionVar_1_1, padding=[1, 1, 1, 1])
/* ty=Tensor[(1, 32, 56, 56), float32] */;
        %9 = nn.bias_add(%8, %FunctionVar_1_2) /* ty=Tensor[(1, 32, 56, 56),
float32] */;
        nn.relu(%9) /* ty=Tensor[(1, 32, 56, 56), float32] */
      } /* ty=fn (Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3),
float32], Tensor[(32), float32]) -> Tensor[(1, 32, 56, 56), float32] */;
      %14 = %13(%10, %11, %12) /* ty=Tensor[(1, 32, 56, 56), float32] */;
      %15 = annotation.compiler_end(%14, compiler="byoc-target") /*
ty=Tensor[(1, 32, 56, 56), float32] */;
      %16 = annotation.compiler_begin(%7, compiler="default") /* ty=int32 */;
      %17 = annotation.compiler_begin(%15, compiler="default") /* ty=Tensor[(1,
32, 56, 56), float32] */;
      %18 = annotation.compiler_begin(%w1, compiler="default") /* ty=Tensor[(32,
32, 3, 3), float32] */;
      %19 = annotation.compiler_begin(%b1, compiler="default") /*
ty=Tensor[(32), float32] */;
      %20 = %while_loop(%16, %17, %18, %19) /* ty=Tensor[(1, 56, 56, 32),
float32] */;
      annotation.compiler_end(%20, compiler="default") /* ty=Tensor[(1, 56, 56,
32), float32] */
    } else {
      %23 = annotation.compiler_begin(%d1, compiler="byoc-target") /*
ty=Tensor[(1, 32, 56, 56), float32] */;
      %24 = annotation.compiler_begin(%w1, compiler="byoc-target") /*
ty=Tensor[(32, 32, 3, 3), float32] */;
      %25 = annotation.compiler_begin(%b1, compiler="byoc-target") /*
ty=Tensor[(32), float32] */;
      %26 = fn (%FunctionVar_0_0: Tensor[(1, 32, 56, 56), float32] /*
ty=Tensor[(1, 32, 56, 56), float32] */, %FunctionVar_0_1: Tensor[(32, 32, 3, 3),
float32] /* ty=Tensor[(32, 32, 3, 3), float32] */, %FunctionVar_0_2:
Tensor[(32), float32] /* ty=Tensor[(32), float32] */,
PartitionedFromPattern="nn.conv2d_nn.bias_add_nn.relu_", Composite="byoc-
target.conv2d_relu_with_bias") -> Tensor[(1, 32, 56, 56), float32] {
        %21 = nn.conv2d(%FunctionVar_0_0, %FunctionVar_0_1, padding=[1, 1, 1,
1]) /* ty=Tensor[(1, 32, 56, 56), float32] */;
        %22 = nn.bias_add(%21, %FunctionVar_0_2) /* ty=Tensor[(1, 32, 56, 56),
float32] */;
        nn.relu(%22) /* ty=Tensor[(1, 32, 56, 56), float32] */
      } /* ty=fn (Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3),
float32], Tensor[(32), float32]) -> Tensor[(1, 32, 56, 56), float32] */;
      %27 = %26(%23, %24, %25) /* ty=Tensor[(1, 32, 56, 56), float32] */;
```

```
      %28 = annotation.compiler_end(%27, compiler="byoc-target") /*
ty=Tensor[(1, 32, 56, 56), float32] */;
      %29 = annotation.compiler_begin(%28, compiler="byoc-target") /*
ty=Tensor[(1, 32, 56, 56), float32] */;
      %30 = reshape(%29, newshape=[1, 56, 56, 32]) /* ty=Tensor[(1, 56, 56, 32),
float32] */;
      annotation.compiler_end(%30, compiler="byoc-target") /* ty=Tensor[(1, 56,
56, 32), float32] */
    }
  } /* ty=fn (int32, Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3),
float32], Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] */;
  %31 = annotation.compiler_begin(0 /* ty=int32 */, compiler="default") /*
ty=int32 */;
  %32 = annotation.compiler_begin(%data, compiler="default") /* ty=Tensor[(1,
32, 56, 56), float32] */;
  %33 = annotation.compiler_begin(%weight, compiler="default") /* ty=Tensor[(32,
32, 3, 3), float32] */;
  %34 = annotation.compiler_begin(%bias, compiler="default") /* ty=Tensor[(32),
float32] */;
  %35 = %while_loop(%31, %32, %33, %34) /* ty=Tensor[(1, 56, 56, 32), float32]
*/;
  annotation.compiler_end(%35, compiler="default") /* ty=Tensor[(1, 56, 56, 32),
float32] */
} /* ty=fn (Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3), float32],
Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] */

[11:02:35] /home/qzylalala/work_space/tvm/src/ir/transform.cc:440: Running pass
AnnotateTargetFunc
[11:02:35] /home/qzylalala/work_space/tvm/src/ir/transform.cc:379: InferType:
Executing module pass with opt level: 0
[11:02:35] /home/qzylalala/work_space/tvm/src/relay/ir/transform.cc:124:
AnnotateTargetFunc: Executing function pass with opt level: 0
[11:02:35] /home/qzylalala/work_space/tvm/src/relay/ir/transform.cc:125:
AnnotateTargetFunc: Input module:
def @main(%data: Tensor[(1, 32, 56, 56), float32] /* ty=Tensor[(1, 32, 56, 56),
float32] */, %weight: Tensor[(32, 32, 3, 3), float32] /* ty=Tensor[(32, 32, 3,
3), float32] */, %bias: Tensor[(32), float32] /* ty=Tensor[(32), float32] */) ->
Tensor[(1, 56, 56, 32), float32] {
  let %while_loop: fn (int32, Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32,
3, 3), float32], Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] /*
ty=fn (int32, Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3), float32],
Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] */ = fn (%iter1:
int32 /* ty=int32 */, %d1: Tensor[(1, 32, 56, 56), float32] /* ty=Tensor[(1, 32,
56, 56), float32] */, %w1: Tensor[(32, 32, 3, 3), float32] /* ty=Tensor[(32, 32,
3, 3), float32] */, %b1: Tensor[(32), float32] /* ty=Tensor[(32), float32] */)
-> Tensor[(1, 56, 56, 32), float32] {
    %0 = less(%iter1, 2 /* ty=int32 */) /* ty=bool */;
    if (%0) {
```

```
      %3 = fn (%FunctionVar_1_0: Tensor[(1, 32, 56, 56), float32] /*
ty=Tensor[(1, 32, 56, 56), float32] */, %FunctionVar_1_1: Tensor[(32, 32, 3, 3),
float32] /* ty=Tensor[(32, 32, 3, 3), float32] */, %FunctionVar_1_2:
Tensor[(32), float32] /* ty=Tensor[(32), float32] */,
PartitionedFromPattern="nn.conv2d_nn.bias_add_nn.relu_", Composite="byoc-
target.conv2d_relu_with_bias") -> Tensor[(1, 32, 56, 56), float32] {
        %1 = nn.conv2d(%FunctionVar_1_0, %FunctionVar_1_1, padding=[1, 1, 1, 1])
/* ty=Tensor[(1, 32, 56, 56), float32] */;
        %2 = nn.bias_add(%1, %FunctionVar_1_2) /* ty=Tensor[(1, 32, 56, 56),
float32] */;
        nn.relu(%2) /* ty=Tensor[(1, 32, 56, 56), float32] */
      } /* ty=fn (Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3),
float32], Tensor[(32), float32]) -> Tensor[(1, 32, 56, 56), float32] */;
      %4 = add(%iter1, 1 /* ty=int32 */) /* ty=int32 */;
      %5 = %3(%d1, %w1, %b1) /* ty=Tensor[(1, 32, 56, 56), float32] */;
      %while_loop(%4, %5, %w1, %b1) /* ty=Tensor[(1, 56, 56, 32), float32] */
    } else {
      %8 = fn (%FunctionVar_0_0: Tensor[(1, 32, 56, 56), float32] /*
ty=Tensor[(1, 32, 56, 56), float32] */, %FunctionVar_0_1: Tensor[(32, 32, 3, 3),
float32] /* ty=Tensor[(32, 32, 3, 3), float32] */, %FunctionVar_0_2:
Tensor[(32), float32] /* ty=Tensor[(32), float32] */,
PartitionedFromPattern="nn.conv2d_nn.bias_add_nn.relu_", Composite="byoc-
target.conv2d_relu_with_bias") -> Tensor[(1, 32, 56, 56), float32] {
        %6 = nn.conv2d(%FunctionVar_0_0, %FunctionVar_0_1, padding=[1, 1, 1, 1])
/* ty=Tensor[(1, 32, 56, 56), float32] */;
        %7 = nn.bias_add(%6, %FunctionVar_0_2) /* ty=Tensor[(1, 32, 56, 56),
float32] */;
        nn.relu(%7) /* ty=Tensor[(1, 32, 56, 56), float32] */
      } /* ty=fn (Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3),
float32], Tensor[(32), float32]) -> Tensor[(1, 32, 56, 56), float32] */;
      %9 = %8(%d1, %w1, %b1) /* ty=Tensor[(1, 32, 56, 56), float32] */;
      reshape(%9, newshape=[1, 56, 56, 32]) /* ty=Tensor[(1, 56, 56, 32),
float32] */
    }
  } /* ty=fn (int32, Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3),
float32], Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] */;
  %while_loop(0 /* ty=int32 */, %data, %weight, %bias) /* ty=Tensor[(1, 56, 56,
32), float32] */
}

[11:02:35] /home/qzylalala/work_space/tvm/src/relay/ir/transform.cc:148:
AnnotateTargetFunc: Output module:
def @main(%data: Tensor[(1, 32, 56, 56), float32] /* ty=Tensor[(1, 32, 56, 56),
float32] */, %weight: Tensor[(32, 32, 3, 3), float32] /* ty=Tensor[(32, 32, 3,
3), float32] */, %bias: Tensor[(32), float32] /* ty=Tensor[(32), float32] */) ->
Tensor[(1, 56, 56, 32), float32] {
  let %while_loop: fn (int32, Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32,
3, 3), float32], Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] /*
```

```
ty=fn (int32, Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3), float32],
Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] */ = fn (%iter1:
int32 /* ty=int32 */, %d1: Tensor[(1, 32, 56, 56), float32] /* ty=Tensor[(1, 32,
56, 56), float32] */, %w1: Tensor[(32, 32, 3, 3), float32] /* ty=Tensor[(32, 32,
3, 3), float32] */, %b1: Tensor[(32), float32] /* ty=Tensor[(32), float32] */)
-> Tensor[(1, 56, 56, 32), float32] {
    %0 = annotation.compiler_begin(%iter1, compiler="default") /* ty=int32 */;
    %1 = annotation.compiler_begin(2 /* ty=int32 */, compiler="default") /*
ty=int32 */;
    %2 = less(%0, %1) /* ty=bool */;
    %3 = annotation.compiler_end(%2, compiler="default") /* ty=bool */;
    if (%3) {
      %4 = annotation.compiler_begin(%iter1, compiler="byoc-target") /* ty=int32
*/;
      %5 = annotation.compiler_begin(1 /* ty=int32 */, compiler="byoc-target")
/* ty=int32 */;
      %6 = add(%4, %5) /* ty=int32 */;
      %7 = annotation.compiler_end(%6, compiler="byoc-target") /* ty=int32 */;
      %10 = annotation.compiler_begin(%d1, compiler="byoc-target") /*
ty=Tensor[(1, 32, 56, 56), float32] */;
      %11 = annotation.compiler_begin(%w1, compiler="byoc-target") /*
ty=Tensor[(32, 32, 3, 3), float32] */;
      %12 = annotation.compiler_begin(%b1, compiler="byoc-target") /*
ty=Tensor[(32), float32] */;
      %13 = fn (%FunctionVar_1_0: Tensor[(1, 32, 56, 56), float32] /*
ty=Tensor[(1, 32, 56, 56), float32] */, %FunctionVar_1_1: Tensor[(32, 32, 3, 3),
float32] /* ty=Tensor[(32, 32, 3, 3), float32] */, %FunctionVar_1_2:
Tensor[(32), float32] /* ty=Tensor[(32), float32] */,
PartitionedFromPattern="nn.conv2d_nn.bias_add_nn.relu_", Composite="byoc-
target.conv2d_relu_with_bias") -> Tensor[(1, 32, 56, 56), float32] {
        %8 = nn.conv2d(%FunctionVar_1_0, %FunctionVar_1_1, padding=[1, 1, 1, 1])
/* ty=Tensor[(1, 32, 56, 56), float32] */;
        %9 = nn.bias_add(%8, %FunctionVar_1_2) /* ty=Tensor[(1, 32, 56, 56),
float32] */;
        nn.relu(%9) /* ty=Tensor[(1, 32, 56, 56), float32] */
      } /* ty=fn (Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3),
float32], Tensor[(32), float32]) -> Tensor[(1, 32, 56, 56), float32] */;
      %14 = %13(%10, %11, %12) /* ty=Tensor[(1, 32, 56, 56), float32] */;
      %15 = annotation.compiler_end(%14, compiler="byoc-target") /*
ty=Tensor[(1, 32, 56, 56), float32] */;
      %16 = annotation.compiler_begin(%7, compiler="default") /* ty=int32 */;
      %17 = annotation.compiler_begin(%15, compiler="default") /* ty=Tensor[(1,
32, 56, 56), float32] */;
      %18 = annotation.compiler_begin(%w1, compiler="default") /* ty=Tensor[(32,
32, 3, 3), float32] */;
      %19 = annotation.compiler_begin(%b1, compiler="default") /*
ty=Tensor[(32), float32] */;
      %20 = %while_loop(%16, %17, %18, %19) /* ty=Tensor[(1, 56, 56, 32),
```

```
float32] */;
      annotation.compiler_end(%20, compiler="default") /* ty=Tensor[(1, 56, 56,
32), float32] */
    } else {
      %23 = annotation.compiler_begin(%d1, compiler="byoc-target") /*
ty=Tensor[(1, 32, 56, 56), float32] */;
      %24 = annotation.compiler_begin(%w1, compiler="byoc-target") /*
ty=Tensor[(32, 32, 3, 3), float32] */;
      %25 = annotation.compiler_begin(%b1, compiler="byoc-target") /*
ty=Tensor[(32), float32] */;
      %26 = fn (%FunctionVar_0_0: Tensor[(1, 32, 56, 56), float32] /*
ty=Tensor[(1, 32, 56, 56), float32] */, %FunctionVar_0_1: Tensor[(32, 32, 3, 3),
float32] /* ty=Tensor[(32, 32, 3, 3), float32] */, %FunctionVar_0_2:
Tensor[(32), float32] /* ty=Tensor[(32), float32] */,
PartitionedFromPattern="nn.conv2d_nn.bias_add_nn.relu_", Composite="byoc-
target.conv2d_relu_with_bias") -> Tensor[(1, 32, 56, 56), float32] {
        %21 = nn.conv2d(%FunctionVar_0_0, %FunctionVar_0_1, padding=[1, 1, 1,
1]) /* ty=Tensor[(1, 32, 56, 56), float32] */;
        %22 = nn.bias_add(%21, %FunctionVar_0_2) /* ty=Tensor[(1, 32, 56, 56),
float32] */;
        nn.relu(%22) /* ty=Tensor[(1, 32, 56, 56), float32] */
      } /* ty=fn (Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3),
float32], Tensor[(32), float32]) -> Tensor[(1, 32, 56, 56), float32] */;
      %27 = %26(%23, %24, %25) /* ty=Tensor[(1, 32, 56, 56), float32] */;
      %28 = annotation.compiler_end(%27, compiler="byoc-target") /*
ty=Tensor[(1, 32, 56, 56), float32] */;
      %29 = annotation.compiler_begin(%28, compiler="byoc-target") /*
ty=Tensor[(1, 32, 56, 56), float32] */;
      %30 = reshape(%29, newshape=[1, 56, 56, 32]) /* ty=Tensor[(1, 56, 56, 32),
float32] */;
      annotation.compiler_end(%30, compiler="byoc-target") /* ty=Tensor[(1, 56,
56, 32), float32] */
    }
  } /* ty=fn (int32, Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3),
float32], Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] */;
  %31 = annotation.compiler_begin(0 /* ty=int32 */, compiler="default") /*
ty=int32 */;
  %32 = annotation.compiler_begin(%data, compiler="default") /* ty=Tensor[(1,
32, 56, 56), float32] */;
  %33 = annotation.compiler_begin(%weight, compiler="default") /* ty=Tensor[(32,
32, 3, 3), float32] */;
  %34 = annotation.compiler_begin(%bias, compiler="default") /* ty=Tensor[(32),
float32] */;
  %35 = %while_loop(%31, %32, %33, %34) /* ty=Tensor[(1, 56, 56, 32), float32]
*/;
  annotation.compiler_end(%35, compiler="default") /* ty=Tensor[(1, 56, 56, 32),
float32] */
}
```

```
[11:02:35] /home/qzylalala/work_space/tvm/src/ir/transform.cc:379:
AnnotateTargetFunc: InferType: Executing module pass with opt level: 0
[11:02:35] /home/qzylalala/work_space/tvm/src/ir/transform.cc:440: Running pass
InferType
[11:02:35] /home/qzylalala/work_space/tvm/src/ir/transform.cc:379: InferType:
Executing module pass with opt level: 0
```

Looks scary! Let me make this Relay graph more readable so that we can easilty find some interesting points.

- Almost all nodes in the graph are annotated with `compiler_begin` and `compiler_end` nodes. `compiler_*` nodes has an attribute `compiler` to indicate which target should this node go. In this example, it can be `default` or `byoc-target`.
- Composite function calls are also annotated with `compiler=byoc-target`, indicating that this entire function can be offloaded.
- We can find that some annotations can actually be merged, such as the annotations for the composite function and the following `reshape`. We use the next pass, `MergeCompilerRegion`, to merge them so that we can minimize the number of subgraphs.

[ ]:
```
mod4 = relay.transform.MergeCompilerRegions()(mod3)
print(mod4["main"].astext(show_meta_data=False))
```

```
#[version = "0.0.5"]
fn (%data: Tensor[(1, 32, 56, 56), float32] /* ty=Tensor[(1, 32, 56, 56),
float32] */, %weight: Tensor[(32, 32, 3, 3), float32] /* ty=Tensor[(32, 32, 3,
3), float32] */, %bias: Tensor[(32), float32] /* ty=Tensor[(32), float32] */) ->
Tensor[(1, 56, 56, 32), float32] {
  let %while_loop: fn (int32, Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32,
3, 3), float32], Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] /*
ty=fn (int32, Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3), float32],
Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] */ = fn (%iter1:
int32 /* ty=int32 */, %d1: Tensor[(1, 32, 56, 56), float32] /* ty=Tensor[(1, 32,
56, 56), float32] */, %w1: Tensor[(32, 32, 3, 3), float32] /* ty=Tensor[(32, 32,
3, 3), float32] */, %b1: Tensor[(32), float32] /* ty=Tensor[(32), float32] */)
-> Tensor[(1, 56, 56, 32), float32] {
    %0 = annotation.compiler_begin(%iter1, compiler="default") /* ty=int32 */;
    %1 = annotation.compiler_begin(2 /* ty=int32 */, compiler="default") /*
ty=int32 */;
    %2 = less(%0, %1) /* ty=bool */;
    %3 = annotation.compiler_end(%2, compiler="default") /* ty=bool */;
    if (%3) {
      %4 = annotation.compiler_begin(%iter1, compiler="byoc-target") /* ty=int32
*/;
      %5 = annotation.compiler_begin(1 /* ty=int32 */, compiler="byoc-target")
/* ty=int32 */;
      %6 = add(%4, %5) /* ty=int32 */;
      %7 = annotation.compiler_end(%6, compiler="byoc-target") /* ty=int32 */;
```

```
      %10 = annotation.compiler_begin(%d1, compiler="byoc-target") /*
ty=Tensor[(1, 32, 56, 56), float32] */;
      %11 = annotation.compiler_begin(%w1, compiler="byoc-target") /*
ty=Tensor[(32, 32, 3, 3), float32] */;
      %12 = annotation.compiler_begin(%b1, compiler="byoc-target") /*
ty=Tensor[(32), float32] */;
      %13 = fn (%FunctionVar_1_0: Tensor[(1, 32, 56, 56), float32] /*
ty=Tensor[(1, 32, 56, 56), float32] */, %FunctionVar_1_1: Tensor[(32, 32, 3, 3),
float32] /* ty=Tensor[(32, 32, 3, 3), float32] */, %FunctionVar_1_2:
Tensor[(32), float32] /* ty=Tensor[(32), float32] */,
PartitionedFromPattern="nn.conv2d_nn.bias_add_nn.relu_", Composite="byoc-
target.conv2d_relu_with_bias") -> Tensor[(1, 32, 56, 56), float32] {
        %8 = nn.conv2d(%FunctionVar_1_0, %FunctionVar_1_1, padding=[1, 1, 1, 1])
/* ty=Tensor[(1, 32, 56, 56), float32] */;
        %9 = nn.bias_add(%8, %FunctionVar_1_2) /* ty=Tensor[(1, 32, 56, 56),
float32] */;
        nn.relu(%9) /* ty=Tensor[(1, 32, 56, 56), float32] */
      } /* ty=fn (Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3),
float32], Tensor[(32), float32]) -> Tensor[(1, 32, 56, 56), float32] */;
      %14 = %13(%10, %11, %12) /* ty=Tensor[(1, 32, 56, 56), float32] */;
      %15 = annotation.compiler_end(%14, compiler="byoc-target") /*
ty=Tensor[(1, 32, 56, 56), float32] */;
      %16 = annotation.compiler_begin(%7, compiler="default") /* ty=int32 */;
      %17 = annotation.compiler_begin(%15, compiler="default") /* ty=Tensor[(1,
32, 56, 56), float32] */;
      %18 = annotation.compiler_begin(%w1, compiler="default") /* ty=Tensor[(32,
32, 3, 3), float32] */;
      %19 = annotation.compiler_begin(%b1, compiler="default") /*
ty=Tensor[(32), float32] */;
      %20 = %while_loop(%16, %17, %18, %19) /* ty=Tensor[(1, 56, 56, 32),
float32] */;
      annotation.compiler_end(%20, compiler="default") /* ty=Tensor[(1, 56, 56,
32), float32] */
    } else {
      %23 = annotation.compiler_begin(%d1, compiler="byoc-target") /*
ty=Tensor[(1, 32, 56, 56), float32] */;
      %24 = annotation.compiler_begin(%w1, compiler="byoc-target") /*
ty=Tensor[(32, 32, 3, 3), float32] */;
      %25 = annotation.compiler_begin(%b1, compiler="byoc-target") /*
ty=Tensor[(32), float32] */;
      %26 = fn (%FunctionVar_0_0: Tensor[(1, 32, 56, 56), float32] /*
ty=Tensor[(1, 32, 56, 56), float32] */, %FunctionVar_0_1: Tensor[(32, 32, 3, 3),
float32] /* ty=Tensor[(32, 32, 3, 3), float32] */, %FunctionVar_0_2:
Tensor[(32), float32] /* ty=Tensor[(32), float32] */,
PartitionedFromPattern="nn.conv2d_nn.bias_add_nn.relu_", Composite="byoc-
target.conv2d_relu_with_bias") -> Tensor[(1, 32, 56, 56), float32] {
        %21 = nn.conv2d(%FunctionVar_0_0, %FunctionVar_0_1, padding=[1, 1, 1,
1]) /* ty=Tensor[(1, 32, 56, 56), float32] */;
```

```
        %22 = nn.bias_add(%21, %FunctionVar_0_2) /* ty=Tensor[(1, 32, 56, 56),
float32] */;
        nn.relu(%22) /* ty=Tensor[(1, 32, 56, 56), float32] */
      } /* ty=fn (Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3),
float32], Tensor[(32), float32]) -> Tensor[(1, 32, 56, 56), float32] */;
      %27 = %26(%23, %24, %25) /* ty=Tensor[(1, 32, 56, 56), float32] */;
      %28 = reshape(%27, newshape=[1, 56, 56, 32]) /* ty=Tensor[(1, 56, 56, 32),
float32] */;
      annotation.compiler_end(%28, compiler="byoc-target") /* ty=Tensor[(1, 56,
56, 32), float32] */
    }
  } /* ty=fn (int32, Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3),
float32], Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] */;
  %29 = annotation.compiler_begin(0 /* ty=int32 */, compiler="default") /*
ty=int32 */;
  %30 = annotation.compiler_begin(%data, compiler="default") /* ty=Tensor[(1,
32, 56, 56), float32] */;
  %31 = annotation.compiler_begin(%weight, compiler="default") /* ty=Tensor[(32,
32, 3, 3), float32] */;
  %32 = annotation.compiler_begin(%bias, compiler="default") /* ty=Tensor[(32),
float32] */;
  %33 = %while_loop(%29, %30, %31, %32) /* ty=Tensor[(1, 56, 56, 32), float32]
*/;
  annotation.compiler_end(%33, compiler="default") /* ty=Tensor[(1, 56, 56, 32),
float32] */
} /* ty=fn (Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3), float32],
Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] */

[11:02:39] /home/qzylalala/work_space/tvm/src/ir/transform.cc:440: Running pass
MergeCompilerRegions
[11:02:39] /home/qzylalala/work_space/tvm/src/relay/ir/transform.cc:124:
MergeCompilerRegions: Executing function pass with opt level: 0
[11:02:39] /home/qzylalala/work_space/tvm/src/relay/ir/transform.cc:125:
MergeCompilerRegions: Input module:
def @main(%data: Tensor[(1, 32, 56, 56), float32] /* ty=Tensor[(1, 32, 56, 56),
float32] */, %weight: Tensor[(32, 32, 3, 3), float32] /* ty=Tensor[(32, 32, 3,
3), float32] */, %bias: Tensor[(32), float32] /* ty=Tensor[(32), float32] */) ->
Tensor[(1, 56, 56, 32), float32] {
  let %while_loop: fn (int32, Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32,
3, 3), float32], Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] /*
ty=fn (int32, Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3), float32],
Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] */ = fn (%iter1:
int32 /* ty=int32 */, %d1: Tensor[(1, 32, 56, 56), float32] /* ty=Tensor[(1, 32,
56, 56), float32] */, %w1: Tensor[(32, 32, 3, 3), float32] /* ty=Tensor[(32, 32,
3, 3), float32] */, %b1: Tensor[(32), float32] /* ty=Tensor[(32), float32] */)
-> Tensor[(1, 56, 56, 32), float32] {
    %0 = annotation.compiler_begin(%iter1, compiler="default") /* ty=int32 */;
    %1 = annotation.compiler_begin(2 /* ty=int32 */, compiler="default") /*
```

```
ty=int32 */;
    %2 = less(%0, %1) /* ty=bool */;
    %3 = annotation.compiler_end(%2, compiler="default") /* ty=bool */;
    if (%3) {
      %4 = annotation.compiler_begin(%iter1, compiler="byoc-target") /* ty=int32
*/;
      %5 = annotation.compiler_begin(1 /* ty=int32 */, compiler="byoc-target")
/* ty=int32 */;
      %6 = add(%4, %5) /* ty=int32 */;
      %7 = annotation.compiler_end(%6, compiler="byoc-target") /* ty=int32 */;
      %10 = annotation.compiler_begin(%d1, compiler="byoc-target") /*
ty=Tensor[(1, 32, 56, 56), float32] */;
      %11 = annotation.compiler_begin(%w1, compiler="byoc-target") /*
ty=Tensor[(32, 32, 3, 3), float32] */;
      %12 = annotation.compiler_begin(%b1, compiler="byoc-target") /*
ty=Tensor[(32), float32] */;
      %13 = fn (%FunctionVar_1_0: Tensor[(1, 32, 56, 56), float32] /*
ty=Tensor[(1, 32, 56, 56), float32] */, %FunctionVar_1_1: Tensor[(32, 32, 3, 3),
float32] /* ty=Tensor[(32, 32, 3, 3), float32] */, %FunctionVar_1_2:
Tensor[(32), float32] /* ty=Tensor[(32), float32] */,
PartitionedFromPattern="nn.conv2d_nn.bias_add_nn.relu_", Composite="byoc-
target.conv2d_relu_with_bias") -> Tensor[(1, 32, 56, 56), float32] {
        %8 = nn.conv2d(%FunctionVar_1_0, %FunctionVar_1_1, padding=[1, 1, 1, 1])
/* ty=Tensor[(1, 32, 56, 56), float32] */;
        %9 = nn.bias_add(%8, %FunctionVar_1_2) /* ty=Tensor[(1, 32, 56, 56),
float32] */;
        nn.relu(%9) /* ty=Tensor[(1, 32, 56, 56), float32] */
      } /* ty=fn (Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3),
float32], Tensor[(32), float32]) -> Tensor[(1, 32, 56, 56), float32] */;
      %14 = %13(%10, %11, %12) /* ty=Tensor[(1, 32, 56, 56), float32] */;
      %15 = annotation.compiler_end(%14, compiler="byoc-target") /*
ty=Tensor[(1, 32, 56, 56), float32] */;
      %16 = annotation.compiler_begin(%7, compiler="default") /* ty=int32 */;
      %17 = annotation.compiler_begin(%15, compiler="default") /* ty=Tensor[(1,
32, 56, 56), float32] */;
      %18 = annotation.compiler_begin(%w1, compiler="default") /* ty=Tensor[(32,
32, 3, 3), float32] */;
      %19 = annotation.compiler_begin(%b1, compiler="default") /*
ty=Tensor[(32), float32] */;
      %20 = %while_loop(%16, %17, %18, %19) /* ty=Tensor[(1, 56, 56, 32),
float32] */;
      annotation.compiler_end(%20, compiler="default") /* ty=Tensor[(1, 56, 56,
32), float32] */
    } else {
      %23 = annotation.compiler_begin(%d1, compiler="byoc-target") /*
ty=Tensor[(1, 32, 56, 56), float32] */;
      %24 = annotation.compiler_begin(%w1, compiler="byoc-target") /*
ty=Tensor[(32, 32, 3, 3), float32] */;
```

```
      %25 = annotation.compiler_begin(%b1, compiler="byoc-target") /*
ty=Tensor[(32), float32] */;
      %26 = fn (%FunctionVar_0_0: Tensor[(1, 32, 56, 56), float32] /*
ty=Tensor[(1, 32, 56, 56), float32] */, %FunctionVar_0_1: Tensor[(32, 32, 3, 3),
float32] /* ty=Tensor[(32, 32, 3, 3), float32] */, %FunctionVar_0_2:
Tensor[(32), float32] /* ty=Tensor[(32), float32] */,
PartitionedFromPattern="nn.conv2d_nn.bias_add_nn.relu_", Composite="byoc-
target.conv2d_relu_with_bias") -> Tensor[(1, 32, 56, 56), float32] {
        %21 = nn.conv2d(%FunctionVar_0_0, %FunctionVar_0_1, padding=[1, 1, 1,
1]) /* ty=Tensor[(1, 32, 56, 56), float32] */;
        %22 = nn.bias_add(%21, %FunctionVar_0_2) /* ty=Tensor[(1, 32, 56, 56),
float32] */;
        nn.relu(%22) /* ty=Tensor[(1, 32, 56, 56), float32] */
      } /* ty=fn (Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3),
float32], Tensor[(32), float32]) -> Tensor[(1, 32, 56, 56), float32] */;
      %27 = %26(%23, %24, %25) /* ty=Tensor[(1, 32, 56, 56), float32] */;
      %28 = annotation.compiler_end(%27, compiler="byoc-target") /*
ty=Tensor[(1, 32, 56, 56), float32] */;
      %29 = annotation.compiler_begin(%28, compiler="byoc-target") /*
ty=Tensor[(1, 32, 56, 56), float32] */;
      %30 = reshape(%29, newshape=[1, 56, 56, 32]) /* ty=Tensor[(1, 56, 56, 32),
float32] */;
      annotation.compiler_end(%30, compiler="byoc-target") /* ty=Tensor[(1, 56,
56, 32), float32] */
    }
  } /* ty=fn (int32, Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3),
float32], Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] */;
  %31 = annotation.compiler_begin(0 /* ty=int32 */, compiler="default") /*
ty=int32 */;
  %32 = annotation.compiler_begin(%data, compiler="default") /* ty=Tensor[(1,
32, 56, 56), float32] */;
  %33 = annotation.compiler_begin(%weight, compiler="default") /* ty=Tensor[(32,
32, 3, 3), float32] */;
  %34 = annotation.compiler_begin(%bias, compiler="default") /* ty=Tensor[(32),
float32] */;
  %35 = %while_loop(%31, %32, %33, %34) /* ty=Tensor[(1, 56, 56, 32), float32]
*/;
  annotation.compiler_end(%35, compiler="default") /* ty=Tensor[(1, 56, 56, 32),
float32] */
}

[11:02:39] /home/qzylalala/work_space/tvm/src/relay/ir/transform.cc:148:
MergeCompilerRegions: Output module:
def @main(%data: Tensor[(1, 32, 56, 56), float32] /* ty=Tensor[(1, 32, 56, 56),
float32] */, %weight: Tensor[(32, 32, 3, 3), float32] /* ty=Tensor[(32, 32, 3,
3), float32] */, %bias: Tensor[(32), float32] /* ty=Tensor[(32), float32] */) ->
Tensor[(1, 56, 56, 32), float32] {
  let %while_loop: fn (int32, Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32,
```

```
3, 3), float32], Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] /*
ty=fn (int32, Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3), float32],
Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] */ = fn (%iter1:
int32 /* ty=int32 */, %d1: Tensor[(1, 32, 56, 56), float32] /* ty=Tensor[(1, 32,
56, 56), float32] */, %w1: Tensor[(32, 32, 3, 3), float32] /* ty=Tensor[(32, 32,
3, 3), float32] */, %b1: Tensor[(32), float32] /* ty=Tensor[(32), float32] */)
-> Tensor[(1, 56, 56, 32), float32] {
    %0 = annotation.compiler_begin(%iter1, compiler="default") /* ty=int32 */;
    %1 = annotation.compiler_begin(2 /* ty=int32 */, compiler="default") /*
ty=int32 */;
    %2 = less(%0, %1) /* ty=bool */;
    %3 = annotation.compiler_end(%2, compiler="default") /* ty=bool */;
    if (%3) {
      %4 = annotation.compiler_begin(%iter1, compiler="byoc-target") /* ty=int32
*/;
      %5 = annotation.compiler_begin(1 /* ty=int32 */, compiler="byoc-target")
/* ty=int32 */;
      %6 = add(%4, %5) /* ty=int32 */;
      %7 = annotation.compiler_end(%6, compiler="byoc-target") /* ty=int32 */;
      %10 = annotation.compiler_begin(%d1, compiler="byoc-target") /*
ty=Tensor[(1, 32, 56, 56), float32] */;
      %11 = annotation.compiler_begin(%w1, compiler="byoc-target") /*
ty=Tensor[(32, 32, 3, 3), float32] */;
      %12 = annotation.compiler_begin(%b1, compiler="byoc-target") /*
ty=Tensor[(32), float32] */;
      %13 = fn (%FunctionVar_1_0: Tensor[(1, 32, 56, 56), float32] /*
ty=Tensor[(1, 32, 56, 56), float32] */, %FunctionVar_1_1: Tensor[(32, 32, 3, 3),
float32] /* ty=Tensor[(32, 32, 3, 3), float32] */, %FunctionVar_1_2:
Tensor[(32), float32] /* ty=Tensor[(32), float32] */,
PartitionedFromPattern="nn.conv2d_nn.bias_add_nn.relu_", Composite="byoc-
target.conv2d_relu_with_bias") -> Tensor[(1, 32, 56, 56), float32] {
        %8 = nn.conv2d(%FunctionVar_1_0, %FunctionVar_1_1, padding=[1, 1, 1, 1])
/* ty=Tensor[(1, 32, 56, 56), float32] */;
        %9 = nn.bias_add(%8, %FunctionVar_1_2) /* ty=Tensor[(1, 32, 56, 56),
float32] */;
        nn.relu(%9) /* ty=Tensor[(1, 32, 56, 56), float32] */
      } /* ty=fn (Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3),
float32], Tensor[(32), float32]) -> Tensor[(1, 32, 56, 56), float32] */;
      %14 = %13(%10, %11, %12) /* ty=Tensor[(1, 32, 56, 56), float32] */;
      %15 = annotation.compiler_end(%14, compiler="byoc-target") /*
ty=Tensor[(1, 32, 56, 56), float32] */;
      %16 = annotation.compiler_begin(%7, compiler="default") /* ty=int32 */;
      %17 = annotation.compiler_begin(%15, compiler="default") /* ty=Tensor[(1,
32, 56, 56), float32] */;
      %18 = annotation.compiler_begin(%w1, compiler="default") /* ty=Tensor[(32,
32, 3, 3), float32] */;
      %19 = annotation.compiler_begin(%b1, compiler="default") /*
ty=Tensor[(32), float32] */;
```

```
      %20 = %while_loop(%16, %17, %18, %19) /* ty=Tensor[(1, 56, 56, 32),
float32] */;
      annotation.compiler_end(%20, compiler="default") /* ty=Tensor[(1, 56, 56,
32), float32] */
    } else {
      %23 = annotation.compiler_begin(%d1, compiler="byoc-target") /*
ty=Tensor[(1, 32, 56, 56), float32] */;
      %24 = annotation.compiler_begin(%w1, compiler="byoc-target") /*
ty=Tensor[(32, 32, 3, 3), float32] */;
      %25 = annotation.compiler_begin(%b1, compiler="byoc-target") /*
ty=Tensor[(32), float32] */;
      %26 = fn (%FunctionVar_0_0: Tensor[(1, 32, 56, 56), float32] /*
ty=Tensor[(1, 32, 56, 56), float32] */, %FunctionVar_0_1: Tensor[(32, 32, 3, 3),
float32] /* ty=Tensor[(32, 32, 3, 3), float32] */, %FunctionVar_0_2:
Tensor[(32), float32] /* ty=Tensor[(32), float32] */,
PartitionedFromPattern="nn.conv2d_nn.bias_add_nn.relu_", Composite="byoc-
target.conv2d_relu_with_bias") -> Tensor[(1, 32, 56, 56), float32] {
        %21 = nn.conv2d(%FunctionVar_0_0, %FunctionVar_0_1, padding=[1, 1, 1,
1]) /* ty=Tensor[(1, 32, 56, 56), float32] */;
        %22 = nn.bias_add(%21, %FunctionVar_0_2) /* ty=Tensor[(1, 32, 56, 56),
float32] */;
        nn.relu(%22) /* ty=Tensor[(1, 32, 56, 56), float32] */
      } /* ty=fn (Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3),
float32], Tensor[(32), float32]) -> Tensor[(1, 32, 56, 56), float32] */;
      %27 = %26(%23, %24, %25) /* ty=Tensor[(1, 32, 56, 56), float32] */;
      %28 = reshape(%27, newshape=[1, 56, 56, 32]) /* ty=Tensor[(1, 56, 56, 32),
float32] */;
      annotation.compiler_end(%28, compiler="byoc-target") /* ty=Tensor[(1, 56,
56, 32), float32] */
    }
  } /* ty=fn (int32, Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3),
float32], Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] */;
  %29 = annotation.compiler_begin(0 /* ty=int32 */, compiler="default") /*
ty=int32 */;
  %30 = annotation.compiler_begin(%data, compiler="default") /* ty=Tensor[(1,
32, 56, 56), float32] */;
  %31 = annotation.compiler_begin(%weight, compiler="default") /* ty=Tensor[(32,
32, 3, 3), float32] */;
  %32 = annotation.compiler_begin(%bias, compiler="default") /* ty=Tensor[(32),
float32] */;
  %33 = %while_loop(%29, %30, %31, %32) /* ty=Tensor[(1, 56, 56, 32), float32]
*/;
  annotation.compiler_end(%33, compiler="default") /* ty=Tensor[(1, 56, 56, 32),
float32] */
}

[11:02:39] /home/qzylalala/work_space/tvm/src/ir/transform.cc:379:
MergeCompilerRegions: InferType: Executing module pass with opt level: 0
```

```
[11:02:39] /home/qzylalala/work_space/tvm/src/ir/transform.cc:440: Running pass
InferType
[11:02:39] /home/qzylalala/work_space/tvm/src/ir/transform.cc:379: InferType:
Executing module pass with opt level: 0
```

We can see that now the `add` and the composite function call in the loop body share the same set of annotation nodes. i.e., the consecutive `compiler_end` and `compiler_begin` nodes are removed.

Finally, let's partition this graph:

[ ]:
```
mod5 = relay.transform.PartitionGraph()(mod4)
print(mod5["main"].astext(show_meta_data=False))
```

```
[11:02:43] /home/qzylalala/work_space/tvm/src/ir/transform.cc:440: Running pass
FlattenNestedTuples
[11:02:43] /home/qzylalala/work_space/tvm/src/ir/transform.cc:379:
FlattenNestedTuples: Executing module pass with opt level: 0
[11:02:43] /home/qzylalala/work_space/tvm/src/ir/transform.cc:379:
FlattenNestedTuples: InferType: Executing module pass with opt level: 0
[11:02:43] /home/qzylalala/work_space/tvm/src/ir/transform.cc:440: Running pass
RemoveDefaultAnnotations
[11:02:43] /home/qzylalala/work_space/tvm/src/ir/transform.cc:379:
RemoveDefaultAnnotations: Executing module pass with opt level: 0
[11:02:43] /home/qzylalala/work_space/tvm/src/ir/transform.cc:379:
RemoveDefaultAnnotations: InferType: Executing module pass with opt level: 0
[11:02:43] /home/qzylalala/work_space/tvm/src/ir/transform.cc:440: Running pass
PartitionGraph
[11:02:43] /home/qzylalala/work_space/tvm/src/ir/transform.cc:379:
PartitionGraph: Executing module pass with opt level: 0
[11:02:43] /home/qzylalala/work_space/tvm/src/ir/transform.cc:379:
PartitionGraph: InferType: Executing module pass with opt level: 0
[11:02:43] /home/qzylalala/work_space/tvm/src/ir/transform.cc:379:
PartitionGraph: InferType: Executing module pass with opt level: 0
[11:02:43] /home/qzylalala/work_space/tvm/src/ir/transform.cc:379:
PartitionGraph: InferType: Executing module pass with opt level: 0
[11:02:43] /home/qzylalala/work_space/tvm/src/ir/transform.cc:379:
PartitionGraph: InferType: Executing module pass with opt level: 0
[11:02:43] /home/qzylalala/work_space/tvm/src/ir/transform.cc:440: Running pass
NameMangleExtFuncs
[11:02:43] /home/qzylalala/work_space/tvm/src/ir/transform.cc:379:
NameMangleExtFuncs: Executing module pass with opt level: 0
[11:02:43] /home/qzylalala/work_space/tvm/src/ir/transform.cc:440: Running pass
InferType
[11:02:43] /home/qzylalala/work_space/tvm/src/ir/transform.cc:379: InferType:
Executing module pass with opt level: 0

#[version = "0.0.5"]
fn (%data: Tensor[(1, 32, 56, 56), float32] /* ty=Tensor[(1, 32, 56, 56),
float32] */, %weight: Tensor[(32, 32, 3, 3), float32] /* ty=Tensor[(32, 32, 3,
```

```
3), float32] */, %bias: Tensor[(32), float32] /* ty=Tensor[(32), float32] */) ->
Tensor[(1, 56, 56, 32), float32] {
  let %while_loop: fn (int32, Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32,
3, 3), float32], Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] /*
ty=fn (int32, Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3), float32],
Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] */ = fn (%iter1:
int32 /* ty=int32 */, %d1: Tensor[(1, 32, 56, 56), float32] /* ty=Tensor[(1, 32,
56, 56), float32] */, %w1: Tensor[(32, 32, 3, 3), float32] /* ty=Tensor[(32, 32,
3, 3), float32] */, %b1: Tensor[(32), float32] /* ty=Tensor[(32), float32] */)
-> Tensor[(1, 56, 56, 32), float32] {
    %0 = less(%iter1, 2 /* ty=int32 */) /* ty=bool */;
    if (%0) {
      %1 = @tvmgen_default_byoc_target_main_0(%iter1) /* ty=int32 */;
      %2 = @tvmgen_default_byoc_target_main_2(%d1, %w1, %b1) /* ty=Tensor[(1,
32, 56, 56), float32] */;
      %while_loop(%1, %2, %w1, %b1) /* ty=Tensor[(1, 56, 56, 32), float32] */
    } else {
      @tvmgen_default_byoc_target_main_5(%d1, %w1, %b1) /* ty=Tensor[(1, 56, 56,
32), float32] */
    }
  } /* ty=fn (int32, Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3),
float32], Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] */;
  %while_loop(0 /* ty=int32 */, %data, %weight, %bias) /* ty=Tensor[(1, 56, 56,
32), float32] */
} /* ty=fn (Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3), float32],
Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] */
```

It's much clean now, right? We can see that 3 subgraphs have been partitioned for `byoc-target`. Let's see dive into each of them:

[ ]:
```
for name in ["tvmgen_default_byoc_target_main_0",␣
↪"tvmgen_default_byoc_target_main_2", "tvmgen_default_byoc_target_main_5"]:
    print("%s: " % name)
    print(mod5[name].astext(show_meta_data=False))
    print("==================")
```

```
tvmgen_default_byoc_target_main_0:
#[version = "0.0.5"]
fn (%byoc-target_0_i0: int32 /* ty=int32 */, Compiler="byoc-target",
Primitive=1, Inline=1, global_symbol="tvmgen_default_byoc_target_main_0") ->
int32 {
  add(%byoc-target_0_i0, 1 /* ty=int32 */) /* ty=int32 */
} /* ty=fn (int32) -> int32 */
==================
tvmgen_default_byoc_target_main_2:
#[version = "0.0.5"]
fn (%byoc-target_2_i0: Tensor[(1, 32, 56, 56), float32] /* ty=Tensor[(1, 32, 56,
56), float32] */, %byoc-target_2_i1: Tensor[(32, 32, 3, 3), float32] /*
```

```
ty=Tensor[(32, 32, 3, 3), float32] */, %byoc-target_2_i2: Tensor[(32), float32]
/* ty=Tensor[(32), float32] */, Compiler="byoc-target", Primitive=1, Inline=1,
global_symbol="tvmgen_default_byoc_target_main_2") -> Tensor[(1, 32, 56, 56),
float32] {
  %2 = fn (%FunctionVar_1_0: Tensor[(1, 32, 56, 56), float32] /* ty=Tensor[(1,
32, 56, 56), float32] */, %FunctionVar_1_1: Tensor[(32, 32, 3, 3), float32] /*
ty=Tensor[(32, 32, 3, 3), float32] */, %FunctionVar_1_2: Tensor[(32), float32]
/* ty=Tensor[(32), float32] */,
PartitionedFromPattern="nn.conv2d_nn.bias_add_nn.relu_", Composite="byoc-
target.conv2d_relu_with_bias") -> Tensor[(1, 32, 56, 56), float32] {
    %0 = nn.conv2d(%FunctionVar_1_0, %FunctionVar_1_1, padding=[1, 1, 1, 1]) /*
ty=Tensor[(1, 32, 56, 56), float32] */;
    %1 = nn.bias_add(%0, %FunctionVar_1_2) /* ty=Tensor[(1, 32, 56, 56),
float32] */;
    nn.relu(%1) /* ty=Tensor[(1, 32, 56, 56), float32] */
  } /* ty=fn (Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3), float32],
Tensor[(32), float32]) -> Tensor[(1, 32, 56, 56), float32] */;
  %2(%byoc-target_2_i0, %byoc-target_2_i1, %byoc-target_2_i2) /* ty=Tensor[(1,
32, 56, 56), float32] */
} /* ty=fn (Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3), float32],
Tensor[(32), float32]) -> Tensor[(1, 32, 56, 56), float32] */
==================
tvmgen_default_byoc_target_main_5:
#[version = "0.0.5"]
fn (%byoc-target_5_i0: Tensor[(1, 32, 56, 56), float32] /* ty=Tensor[(1, 32, 56,
56), float32] */, %byoc-target_5_i1: Tensor[(32, 32, 3, 3), float32] /*
ty=Tensor[(32, 32, 3, 3), float32] */, %byoc-target_5_i2: Tensor[(32), float32]
/* ty=Tensor[(32), float32] */, Compiler="byoc-target", Primitive=1, Inline=1,
global_symbol="tvmgen_default_byoc_target_main_5") -> Tensor[(1, 56, 56, 32),
float32] {
  %2 = fn (%FunctionVar_0_0: Tensor[(1, 32, 56, 56), float32] /* ty=Tensor[(1,
32, 56, 56), float32] */, %FunctionVar_0_1: Tensor[(32, 32, 3, 3), float32] /*
ty=Tensor[(32, 32, 3, 3), float32] */, %FunctionVar_0_2: Tensor[(32), float32]
/* ty=Tensor[(32), float32] */,
PartitionedFromPattern="nn.conv2d_nn.bias_add_nn.relu_", Composite="byoc-
target.conv2d_relu_with_bias") -> Tensor[(1, 32, 56, 56), float32] {
    %0 = nn.conv2d(%FunctionVar_0_0, %FunctionVar_0_1, padding=[1, 1, 1, 1]) /*
ty=Tensor[(1, 32, 56, 56), float32] */;
    %1 = nn.bias_add(%0, %FunctionVar_0_2) /* ty=Tensor[(1, 32, 56, 56),
float32] */;
    nn.relu(%1) /* ty=Tensor[(1, 32, 56, 56), float32] */
  } /* ty=fn (Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3), float32],
Tensor[(32), float32]) -> Tensor[(1, 32, 56, 56), float32] */;
  %3 = %2(%byoc-target_5_i0, %byoc-target_5_i1, %byoc-target_5_i2) /*
ty=Tensor[(1, 32, 56, 56), float32] */;
  reshape(%3, newshape=[1, 56, 56, 32]) /* ty=Tensor[(1, 56, 56, 32), float32]
*/
} /* ty=fn (Tensor[(1, 32, 56, 56), float32], Tensor[(32, 32, 3, 3), float32],
```

```
Tensor[(32), float32]) -> Tensor[(1, 56, 56, 32), float32] */
==================
```

Each partitioned function will be sent to the "byoc-target" codegen for code generation. As a result, you can imagine that the customized codegen only needs to consider the subgraphs without worrying about rest parts of the graph. In this example, it also means that the customzied codegen doesn't have to worry abou the control flow (nice!

In the rest part of this demo, we are going to build a real world SSD model with the TensorRT BYOC integration, which is already available in the upstream TVM so you are welcome to try it by yourself. Specifically, we will build a Gluon CV SSD-ResNet50 model with TensorRT.

Please note that in order to run this example by yourself, you need to set up TensorRT in your environment and build the TVM with TensorRT runtime. You can refer to the TVM TensorRT tutorial for detail instructions: https://tvm.apache.org/docs/deploy/tensorrt.html