HW 3: TVM!

2021-01 인공지능 플랫폼 최적화
HW/SW Optimization for Machine Learning
박영준

Hanyang University
Division of Computer Science & Engineering

HW #3: TVM Programming & Optimization

- 과제: 본 문서의 Step 1 ~ Step 5 의 내용을 직접 수행하고 결과를 레포트로 제출. 레포트에는 다음의 내용이 필수적으로 포함되어야 함
  - [Step 1] 연산 결과 및 네트워크 파일 이름이 출력된 스크린 샷
  - [Step 2] 각 연산 최적화 방법이 적용된 실행 결과 스크린 샷
  - [Step 3] Convolution 연산 시간이 출력된 스크린 샷
  - [Step 4] Relay Pass를 활용한 결과가 포함된 스크린 샷
  - [Step 5] VTA 연산의 최적화 결과가 포함된 스크린 샷
- Deadline: 6/4 (Friday) 23:59:59
- Format: PDF or MS word file
- 문의: 정선욱 (metaljsw2@hanyang.ac.kr)

TVM

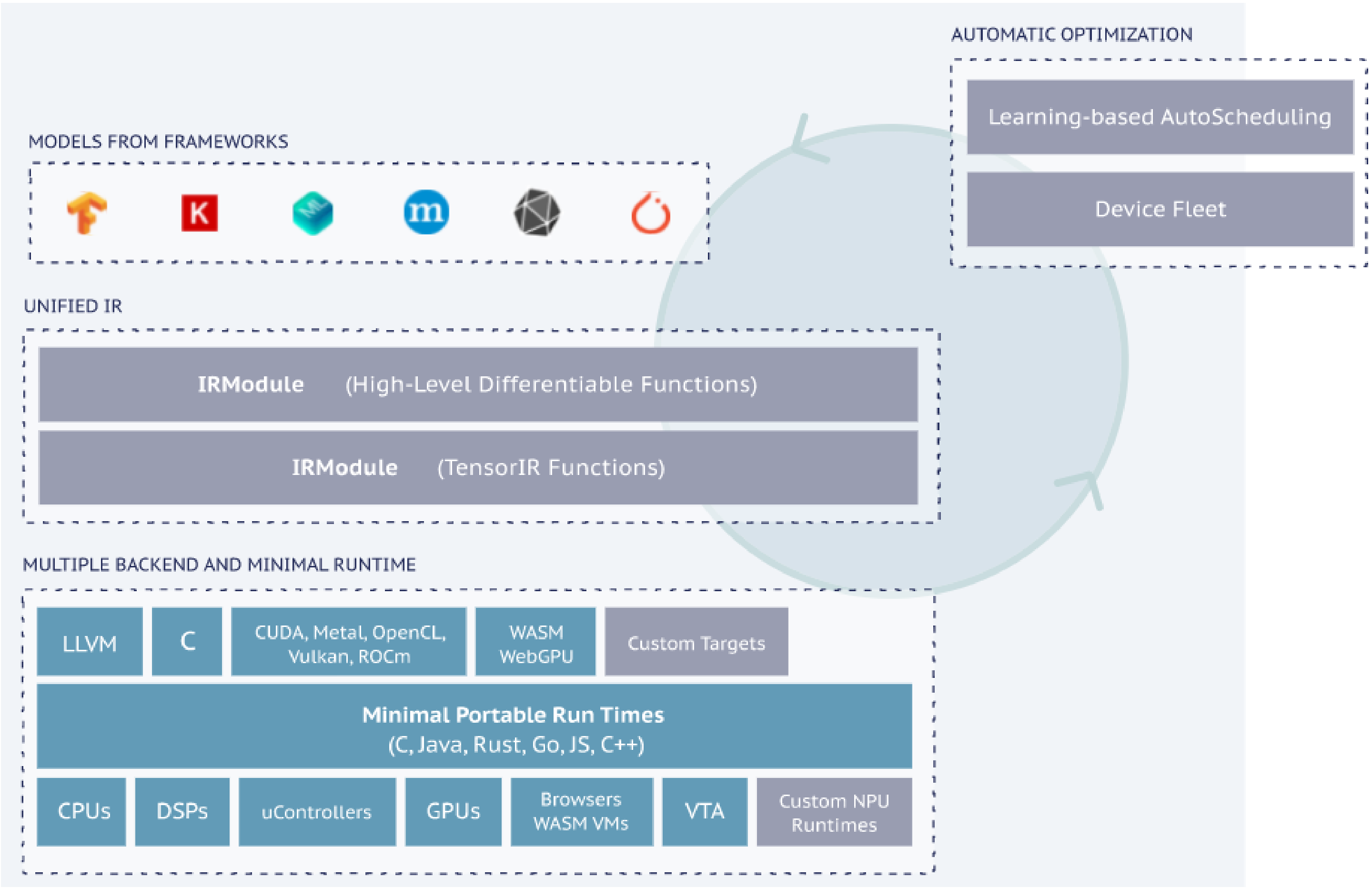
- TVM
  - 자동화된 딥러닝 최적화 프레임워크 Stack
  - CPU, GPU, NPU, FPGA 등의 다양한 Architecture에 적용 가능

- TVM의 특징
  - Python 스크립트로 되어 있어 비교적 쉬운 프로그래밍
  - Tensorflow, pytorch 등의 Framework에서 생성된 모델의 TVM 적용 가능
  - pre-build된 모듈을 동일한 Architecture의 다른 디바이스에서 사용가능

- TVM의 한계
  - 오픈소스이기 때문에 지속적으로 기능 변경이 있다.
  - 현재 Training 이 불가능하다.

Hanyang University
Division of Computer Science & Engineering

TVM



TVM paper: https://arxiv.org/pdf/1802.04799.pdf

TVM: Installation

- Install from Source
  - 아래의 Github 주소에서 TVM 소스코드를 다운

    https://github.com/apache/tvm
  - 설치 안내를 바탕으로 소스코드를 빌드

    https://tvm.apache.org/docs/install/from_source.html

[Step 1] Getting Started

- Quick Start Tutorial for Compiling Deep Learning Models
  - 이 단계에서는 Relay python frontend로 Neural Network를 구축하고 TVM을 사용하는 NVIDIA GPU 용 런타임 라이브러리를 생성하는 방법을 수행합니다.
  - VGG16 모델을 TVM으로 컴파일 해보고 생성되는 모델 코드와 결과값을 확인해볼 수 있습니다.
- Pre-requisite
  - CUDA 및 LLVM 설치가 반드시 필요합니다.

  Reference: Installing a CUDA on Ubuntu OS:

  URL: https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html

  Reference: Installing a llvm with Source Code (recommend version: 8.0.0)

  Source code: https://releases.llvm.org/download.html
- Baseline
  - 이번 단계의 Baseline을 아래의 링크에서 확인할 수 있습니다.

  URL: https://tvm.apache.org/docs/tutorials/get_started/relay_quick_start.html

[Step 1] Getting Started

- Quick Start Tutorial for Compiling Deep Learning Models
  - vgg16.py

```python
import numpy as np

from tvm import relay
from tvm.relay import testing
import tvm
from tvm import te
from tvm.contrib import graph_executor
import tvm.testing

batch_size = 1
num_class = 1000
image_shape = (3, 224, 224)
data_shape = (batch_size,) + image_shape
out_shape = (batch_size, num_class)

mod, params = relay.testing.vgg.get_workload(
    num_layers=16, batch_size=batch_size, image_shape=image_shape
)

opt_level = 3
target = tvm.target.cuda()
with tvm.transform.PassContext(opt_level=opt_level):
    lib = relay.build(mod, target, params=params)

# create random input
dev = tvm.cuda()
data = np.random.uniform(-1, 1, size=data_shape).astype("float32")
# create module
module = graph_executor.GraphModule(lib["default"](dev))
# set input and parameters
module.set_input("data", data)
```

```python
# run
module.run()
# get output
out = module.get_output(0, tvm.nd.empty(out_shape)).asnumpy()

# Print first 10 elements of output
print(out.flatten()[0:10])

# save the graph, lib and params into separate files
from tvm.contrib import utils

temp = utils.tempdir()
path_lib = temp.relpath("deploy_lib.tar")
lib.export_library(path_lib)
print(temp.listdir())

# load the module back.
loaded_lib = tvm.runtime.load_module(path_lib)
input_data = tvm.nd.array(data)

module = graph_executor.GraphModule(loaded_lib["default"](dev))
module.run(data=input_data)
out_deploy = module.get_output(0).asnumpy()

# Print first 10 elements of output
print(out_deploy.flatten()[0:10])

# check whether the output from deployed module is consistent with original one
tvm.testing.assert_allclose(out_deploy, out, atol=1e-5)
```

[Step 1] Getting Started

- Quick Start Tutorial for Compiling Deep Learning Models
  - 실행 결과 출력부분

```python
import numpy as np

from tvm import relay
from tvm.relay import testing
import tvm
from tvm import te
from tvm.contrib import graph_executor
import tvm.testing

batch_size = 1
num_class = 1000
image_shape = (3, 224, 224)
data_shape = (batch_size,) + image_shape
out_shape = (batch_size, num_class)

mod, params = relay.testing.vgg.get_workload(
    num_layers=16, batch_size=batch_size, image_shape=image_shape
)

opt_level = 3
target = tvm.target.cuda()
with tvm.transform.PassContext(opt_level=opt_level):
    lib = relay.build(mod, target, params=params)

# create random input
dev = tvm.cuda()
data = np.random.uniform(-1, 1, size=data_shape).astype("float32")
# create module
module = graph_executor.GraphModule(lib["default"](dev))
# set input and parameters
module.set_input("data", data)
```

```python
# run
module.run()
# get output
out = module.get_output(0, tvm.nd.empty(out_shape)).asnumpy()

# Print first 10 elements of output
print(out.flatten()[0:10])

# save the graph, lib and params into separate files
from tvm.contrib import utils

temp = utils.tempdir()
path_lib = temp.relpath("deploy_lib.tar")
lib.export_library(path_lib)
print(temp.listdir())

# load the module back.
loaded_lib = tvm.runtime.load_module(path_lib)
input_data = tvm.nd.array(data)

module = graph_executor.GraphModule(loaded_lib["default"](dev))
module.run(data=input_data)
out_deploy = module.get_output(0).asnumpy()

# Print first 10 elements of output
print(out_deploy.flatten()[0:10])

# check whether the output from deployed module is consistent with original one
tvm.testing.assert_allclose(out_deploy, out, atol=1e-5)
```


Hanyang University
Division of Computer Science & Engineering

[Step 1] Getting Started

- Quick Start Tutorial for Compiling Deep Learning Models
  - **아래와 실행 결과를 screenshot으로 제출**

(1) 연산 결과가 올바르게 출력 되는지 (직접 수행한 결과가 아래와 동일한 숫자일 필요는 없습니다.)

```
[0.00100052 0.00099585 0.00099843 0.00100801 0.00099786 0.00100267
 0.00100183 0.00100203 0.00099908 0.0010008 ]
```

(2) 올바르게 저장된 네트워크 파일 이름과 이를 load한 후 위와 동일한 연산 결과가 나오는지

```
['deploy_lib.tar']
[0.00100052 0.00099585 0.00099843 0.00100801 0.00099786 0.00100267
 0.00100183 0.00100203 0.00099908 0.0010008 ]
```

[Step 2] Tensor Expressions and Schedules

- Schedule Primitives in TVM
  - 동일한 결과를 도출하는 연산 방법에는 여러가지가 있지만, 그 방법에 따라 **성능**이 달라지기 때문에 TVM을 이용하여 scheduling을 수행해야 합니다.
  - 이번 단계에서는 TVM에서 제공하는 다양한 primitive로 연속적인 연산을 scheduling 하는 방법을 수행합니다.
  - 이번 단계에서 사용하는 primitive는 **split, tile, fuse, reorder, bind** 총 5가지를 사용합니다.
- Baseline
  - 이번 단계의 Baseline을 아래의 링크에서 확인할 수 있습니다.

  URL: https://tvm.apache.org/docs/tutorials/language/schedule_primitives.html

[Step 2] Tensor Expressions and Schedules

- Schedule Primitives in TVM
  - schedule_primitive.py

```
from __future__ import absolute_import, print_function

import tvm
from tvm import te
import numpy as np

# declare some variables for use later
n = te.var("n")
m = te.var("m")

# declare a matrix element-wise multiply
A = te.placeholder((m, n), name="A")
B = te.placeholder((m, n), name="B")
C = te.compute((m, n), lambda i, j: A[i, j] * B[i, j], name="C")

s = te.create_schedule([C.op])
# lower will transform the computation from definition to the real
# callable function. With argument `simple_mode=True`, it will
# return you a readable C like statement, we use it here to print the
# schedule result.
print(tvm.lower(s, [A, B, C], simple_mode=True))

A = te.placeholder((m,), name="A")
B = te.compute((m,), lambda i: A[i] * 2, name="B")

s = te.create_schedule(B.op)
xo, xi = s[B].split(B.op.axis[0], factor=32)
print(tvm.lower(s, [A, B], simple_mode=True))

A = te.placeholder((m,), name="A")
B = te.compute((m,), lambda i: A[i], name="B")

s = te.create_schedule(B.op)
bx, tx = s[B].split(B.op.axis[0], nparts=32)
print(tvm.lower(s, [A, B], simple_mode=True))
```

```
A = te.placeholder((m, n), name="A")
B = te.compute((m, n), lambda i, j: A[i, j], name="B")

s = te.create_schedule(B.op)
xo, yo, xi, yi = s[B].tile(B.op.axis[0], B.op.axis[1], x_factor=10, y_factor=5)
print(tvm.lower(s, [A, B], simple_mode=True))

A = te.placeholder((m, n), name="A")
B = te.compute((m, n), lambda i, j: A[i, j], name="B")

s = te.create_schedule(B.op)
# tile to four axes first: (i.outer, j.outer, i.inner, j.inner)
xo, yo, xi, yi = s[B].tile(B.op.axis[0], B.op.axis[1], x_factor=10, y_factor=5)
# then fuse (i.inner, j.inner) into one axis: (i.inner.j.inner.fused)
fused = s[B].fuse(xi, yi)
print(tvm.lower(s, [A, B], simple_mode=True))

A = te.placeholder((m, n), name="A")
B = te.compute((m, n), lambda i, j: A[i, j], name="B")

s = te.create_schedule(B.op)
# tile to four axes first: (i.outer, j.outer, i.inner, j.inner)
xo, yo, xi, yi = s[B].tile(B.op.axis[0], B.op.axis[1], x_factor=10, y_factor=5)
# then reorder the axes: (i.inner, j.outer, i.outer, j.inner)
s[B].reorder(xi, yo, xo, yi)
print(tvm.lower(s, [A, B], simple_mode=True))

A = te.placeholder((n,), name="A")
B = te.compute(A.shape, lambda i: A[i] * 2, name="B")

s = te.create_schedule(B.op)
bx, tx = s[B].split(B.op.axis[0], factor=64)
s[B].bind(bx, te.thread_axis("blockIdx.x"))
s[B].bind(tx, te.thread_axis("threadIdx.x"))
print(tvm.lower(s, [A, B], simple_mode=True))
```


Hanyang University
Division of Computer Science & Engineering

[Step 2] Tensor Expressions and Schedules

- Schedule Primitives in TVM
  - Matrix multiply가 정의된 부분

```
from __future__ import absolute_import, print_function

import tvm
from tvm import te
import numpy as np

# declare some variables for use later
n = te.var("n")
m = te.var("m")

# declare a matrix element-wise multiply
A = te.placeholder((m, n), name="A")
B = te.placeholder((m, n), name="B")
C = te.compute((m, n), lambda i, j: A[i, j] * B[i, j], name="C")

s = te.create_schedule([C.op])
# lower will transform the computation from definition to the real
# callable function. With argument `simple_mode=True`, it will
# return you a readable C like statement, we use it here to print the
# schedule result.
print(tvm.lower(s, [A, B, C], simple_mode=True))

A = te.placeholder((m,), name="A")
B = te.compute((m,), lambda i: A[i] * 2, name="B")

s = te.create_schedule(B.op)
xo, xi = s[B].split(B.op.axis[0], factor=32)
print(tvm.lower(s, [A, B], simple_mode=True))

A = te.placeholder((m,), name="A")
B = te.compute((m,), lambda i: A[i], name="B")

s = te.create_schedule(B.op)
bx, tx = s[B].split(B.op.axis[0], nparts=32)
print(tvm.lower(s, [A, B], simple_mode=True))
```

```
A = te.placeholder((m, n), name="A")
B = te.compute((m, n), lambda i, j: A[i, j], name="B")

s = te.create_schedule(B.op)
xo, yo, xi, yi = s[B].tile(B.op.axis[0], B.op.axis[1], x_factor=10, y_factor=5)
print(tvm.lower(s, [A, B], simple_mode=True))

A = te.placeholder((m, n), name="A")
B = te.compute((m, n), lambda i, j: A[i, j], name="B")

s = te.create_schedule(B.op)
# tile to four axes first: (i.outer, j.outer, i.inner, j.inner)
xo, yo, xi, yi = s[B].tile(B.op.axis[0], B.op.axis[1], x_factor=10, y_factor=5)
# then fuse (i.inner, j.inner) into one axis: (i.inner.j.inner.fused)
fused = s[B].fuse(xi, yi)
print(tvm.lower(s, [A, B], simple_mode=True))

A = te.placeholder((m, n), name="A")
B = te.compute((m, n), lambda i, j: A[i, j], name="B")

s = te.create_schedule(B.op)
# tile to four axes first: (i.outer, j.outer, i.inner, j.inner)
xo, yo, xi, yi = s[B].tile(B.op.axis[0], B.op.axis[1], x_factor=10, y_factor=5)
# then reorder the axes: (i.inner, j.outer, i.outer, j.inner)
s[B].reorder(xi, yo, xo, yi)
print(tvm.lower(s, [A, B], simple_mode=True))

A = te.placeholder((n,), name="A")
B = te.compute(A.shape, lambda i: A[i] * 2, name="B")

s = te.create_schedule(B.op)
bx, tx = s[B].split(B.op.axis[0], factor=64)
s[B].bind(bx, te.thread_axis("blockIdx.x"))
s[B].bind(tx, te.thread_axis("threadIdx.x"))
print(tvm.lower(s, [A, B], simple_mode=True))
```

[Step 2] Tensor Expressions and Schedules

- Schedule Primitives in TVM
  - 실행결과 출력 부분

```python
from __future__ import absolute_import, print_function

import tvm
from tvm import te
import numpy as np

# declare some variables for use later
n = te.var("n")
m = te.var("m")

# declare a matrix element-wise multiply
A = te.placeholder((m, n), name="A")
B = te.placeholder((m, n), name="B")
C = te.compute((m, n), lambda i, j: A[i, j] * B[i, j], name="C")

s = te.create_schedule([C.op])
# lower will transform the computation from definition to the real
# callable function. With argument `simple_mode=True`, it will
# return you a readable C like statement, we use it here to print the
# schedule result.
print(tvm.lower(s, [A, B, C], simple_mode=True))

A = te.placeholder((m,), name="A")
B = te.compute((m,), lambda i: A[i] * 2, name="B")

s = te.create_schedule(B.op)
xo, xi = s[B].split(B.op.axis[0], factor=32)
print(tvm.lower(s, [A, B], simple_mode=True))

A = te.placeholder((m,), name="A")
B = te.compute((m,), lambda i: A[i], name="B")

s = te.create_schedule(B.op)
bx, tx = s[B].split(B.op.axis[0], nparts=32)
print(tvm.lower(s, [A, B], simple_mode=True))
```

```python
A = te.placeholder((m, n), name="A")
B = te.compute((m, n), lambda i, j: A[i, j], name="B")

s = te.create_schedule(B.op)
xo, yo, xi, yi = s[B].tile(B.op.axis[0], B.op.axis[1], x_factor=10, y_factor=5)
print(tvm.lower(s, [A, B], simple_mode=True))

A = te.placeholder((m, n), name="A")
B = te.compute((m, n), lambda i, j: A[i, j], name="B")

s = te.create_schedule(B.op)
# tile to four axes first: (i.outer, j.outer, i.inner, j.inner)
xo, yo, xi, yi = s[B].tile(B.op.axis[0], B.op.axis[1], x_factor=10, y_factor=5)
# then fuse (i.inner, j.inner) into one axis: (i.inner.j.inner.fused)
fused = s[B].fuse(xi, yi)
print(tvm.lower(s, [A, B], simple_mode=True))

A = te.placeholder((m, n), name="A")
B = te.compute((m, n), lambda i, j: A[i, j], name="B")

s = te.create_schedule(B.op)
# tile to four axes first: (i.outer, j.outer, i.inner, j.inner)
xo, yo, xi, yi = s[B].tile(B.op.axis[0], B.op.axis[1], x_factor=10, y_factor=5)
# then reorder the axes: (i.inner, j.outer, i.outer, j.inner)
s[B].reorder(xi, yo, xo, yi)
print(tvm.lower(s, [A, B], simple_mode=True))

A = te.placeholder((n,), name="A")
B = te.compute(A.shape, lambda i: A[i] * 2, name="B")

s = te.create_schedule(B.op)
bx, tx = s[B].split(B.op.axis[0], factor=64)
s[B].bind(bx, te.thread_axis("blockIdx.x"))
s[B].bind(tx, te.thread_axis("threadIdx.x"))
print(tvm.lower(s, [A, B], simple_mode=True))
```


Hanyang University
Division of Computer Science & Engineering

[Step 2] Tensor Expressions and Schedules

- Schedule Primitives in TVM
  - 아래의 실행 결과를 Screenshot으로 제출

(1) Matrix Multiply Default

```
primfn(A_1: handle, B_1: handle, C_1: handle) -> ()
  attr = {"global_symbol": "main", "tir.noalias": True}
  buffers = {C: Buffer(C_2: Pointer(float32), float32, [m: int32, n: int32], [stride: int32, stride_1: int32], type="auto"),
             A: Buffer(A_2: Pointer(float32), float32, [m, n], [stride_2: int32, stride_3: int32], type="auto"),
             B: Buffer(B_2: Pointer(float32), float32, [m, n], [stride_4: int32, stride_5: int32], type="auto")}
  buffer_map = {A_1: A, B_1: B, C_1: C} {
  for (i: int32, 0, m) {
    for (j: int32, 0, n) {
      C_2[((i*stride) + (j*stride_1))] = ((float32*)A_2[((i*stride_2) + (j*stride_3))]*(float32*)B_2[((i*stride_4) + (j*stride_5))])
    }
  }
}
```

[Step 2] Tensor Expressions and Schedules

- Schedule Primitives in TVM
  - **아래의 실행 결과를 screenshot으로 제출**

(2-1) Split with factor

```
primfn(A_1: handle, B_1: handle) -> ()
  attr = {"global_symbol": "main", "tir.noalias": True}
  buffers = {B: Buffer(B_2: Pointer(float32), float32, [m: int32], [stride: int32], type="auto"),
             A: Buffer(A_2: Pointer(float32), float32, [m], [stride_1: int32], type="auto")}
  buffer_map = {A_1: A, B_1: B} {
  for (i.outer: int32, 0, floordiv((m + 31), 32)) {
    for (i.inner: int32, 0, 32) {
      if @tir.likely((((i.outer*32) + i.inner) < m), dtype=bool) {
        B_2[(((i.outer*32) + i.inner)*stride)] = ((float32*)A_2[(((i.outer*32) + i.inner)*stride_1)]*2f32)
      }
    }
  }
}
```

(2-2) Split with nparts

```
primfn(A_1: handle, B_1: handle) -> ()
  attr = {"global_symbol": "main", "tir.noalias": True}
  buffers = {B: Buffer(B_2: Pointer(float32), float32, [m: int32], [stride: int32], type="auto"),
             A: Buffer(A_2: Pointer(float32), float32, [m], [stride_1: int32], type="auto")}
  buffer_map = {A_1: A, B_1: B} {
  for (i.outer: int32, 0, 32) {
    for (i.inner: int32, 0, floordiv((m + 31), 32)) {
      if @tir.likely(((i.inner + (i.outer*floordiv((m + 31), 32))) < m), dtype=bool) {
        B_2[((i.inner + (i.outer*floordiv((m + 31), 32)))*stride)] = (float32*)A_2[((i.inner + (i.outer*floordiv((m + 31), 32)))*stride_1)]
      }
    }
  }
}
```

[Step 2] Tensor Expressions and Schedules

- Schedule Primitives in TVM
  - **아래의 실행 결과를 screenshot으로 제출**

(3) Tile

```
primfn(A_1: handle, B_1: handle) -> ()
  attr = {"global_symbol": "main", "tir.noalias": True}
  buffers = {B: Buffer(B_2: Pointer(float32), float32, [m: int32, n: int32], [stride: int32, stride_1: int32], type="auto"),
             A: Buffer(A_2: Pointer(float32), float32, [m, n], [stride_2: int32, stride_3: int32], type="auto")}
  buffer_map = {A_1: A, B_1: B} {
  for (i.outer: int32, 0, floordiv((m + 9), 10)) {
    for (j.outer: int32, 0, floordiv((n + 4), 5)) {
      for (i.inner: int32, 0, 10) {
        if @tir.likely((((i.outer*10) + i.inner) < m), dtype=bool) {
          for (j.inner: int32, 0, 5) {
            if @tir.likely((((j.outer*5) + j.inner) < n), dtype=bool) {
              B_2[((((i.outer*10) + i.inner)*stride) + (((j.outer*5) + j.inner)*stride_1))] = (float32*)A_2[((((i.outer*10) + i.inner)*stride_2) + (((j.outer*5) + j.inner)*stride_3))]
            }
          }
        }
      }
    }
  }
}
```

(4) Fuse

```
primfn(A_1: handle, B_1: handle) -> ()
  attr = {"global_symbol": "main", "tir.noalias": True}
  buffers = {B: Buffer(B_2: Pointer(float32), float32, [m: int32, n: int32], [stride: int32, stride_1: int32], type="auto"),
             A: Buffer(A_2: Pointer(float32), float32, [m, n], [stride_2: int32, stride_3: int32], type="auto")}
  buffer_map = {A_1: A, B_1: B} {
  for (i.outer: int32, 0, floordiv((m + 9), 10)) {
    for (j.outer: int32, 0, floordiv((n + 4), 5)) {
      for (i.inner.j.inner.fused: int32, 0, 50) {
        if @tir.likely((((i.outer*10) + floordiv(i.inner.j.inner.fused, 5)) < m), dtype=bool) {
          if @tir.likely((((j.outer*5) + floormod(i.inner.j.inner.fused, 5)) < n), dtype=bool) {
            B_2[((((i.outer*10) + floordiv(i.inner.j.inner.fused, 5))*stride) + (((j.outer*5) + floormod(i.inner.j.inner.fused, 5))*stride_1))] = (float32*)A_2[((((i.outer*10) + floordiv(i.inner.j.inner.fused, 5))*stride_2) + (((j.outer*5) + floormod(i.inner.j.i
nner.fused, 5))*stride_3))]
          }
        }
      }
    }
  }
}
```


Hanyang University
Division of Computer Science & Engineering

[Step 2] Tensor Expressions and Schedules

- Schedule Primitives in TVM
  - **아래의 실행 결과를 screenshot으로 제출**

(5) Reorder

```
primfn(A_1: handle, B_1: handle) -> ()
  attr = {"global_symbol": "main", "tir.noalias": True}
  buffers = {B: Buffer(B_2: Pointer(float32), float32, [m: int32, n: int32], [stride: int32, stride_1: int32], type="auto"),
             A: Buffer(A_2: Pointer(float32), float32, [m, n], [stride_2: int32, stride_3: int32], type="auto")}
  buffer_map = {A_1: A, B_1: B} {
  for (i.inner: int32, 0, 10) {
    for (j.outer: int32, 0, floordiv((n + 4), 5)) {
      for (i.outer: int32, 0, floordiv((m + 9), 10)) {
        if @tir.likely((((i.outer*10) + i.inner) < m), dtype=bool) {
          for (j.inner: int32, 0, 5) {
            if @tir.likely((((j.outer*5) + j.inner) < n), dtype=bool) {
              B_2[((((i.outer*10) + i.inner)*stride) + (((j.outer*5) + j.inner)*stride_1))] = (float32*)A_2[((((i.outer*10) + i.inner)*stride_2) + (((j.outer*5) + j.inner)*stride_3))]
            }
          }
        }
      }
    }
  }
}
```

(6) Bind

```
primfn(A_1: handle, B_1: handle) -> ()
  attr = {"global_symbol": "main", "tir.noalias": True}
  buffers = {B: Buffer(B_2: Pointer(float32), float32, [n: int32], [stride: int32], type="auto"),
             A: Buffer(A_2: Pointer(float32), float32, [n], [stride_1: int32], type="auto")}
  buffer_map = {A_1: A, B_1: B} {
  attr [IterVar(blockIdx.x: int32, (nullptr), "ThreadIndex", "blockIdx.x")] "thread_extent" = floordiv((n + 63), 64);
  attr [IterVar(threadIdx.x: int32, (nullptr), "ThreadIndex", "threadIdx.x")] "thread_extent" = 64;
  if @tir.likely((((blockIdx.x*64) + threadIdx.x) < n), dtype=bool) {
    B_2[(((blockIdx.x*64) + threadIdx.x)*stride)] = ((float32*)A_2[(((blockIdx.x*64) + threadIdx.x)*stride_1)]*2f32)
  }
}
```


Hanyang University
Division of Computer Science & Engineering

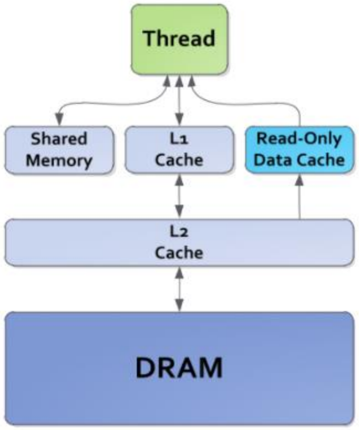
[Step 3] Optimization Tensor Operators

- Optimize Convolution on GPU

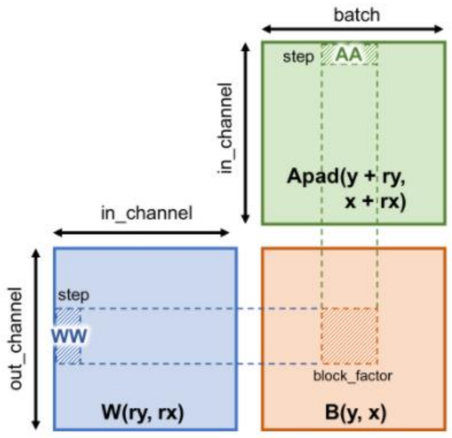
  해당 단계에서는 사각형 형태의 Convolution의 Input Tensor와 Filter Tensor를 사용해서 GPU CUDA 환경에서의 연산을 수행한다. 또한 아래와 같이,

  - GPU 공유 메모리를 활용한 Buffer Caching을 수행하는 Memory Hierarchy,
  - Thread Block 단위로 Split를 수행하는 Blocking,
  - Thread Block을 개별 Thread 단위로 나누는 Virtual thread Split,
  - GPU의 global memory 위의 Data를 shared memory로 전달하도록 변경하는 Cooperative Fetching

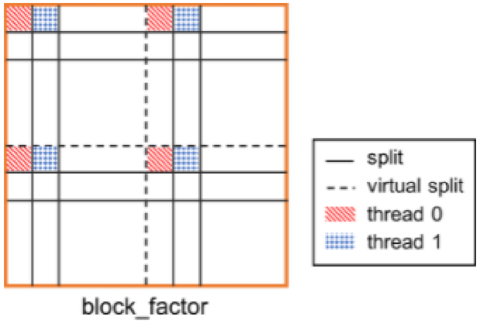
  등의 최적화 방법을 수행해본다.



Memory Hierarchy



Blocking



Virtual thread Split

[Step 3] Optimization Tensor Operators

- Baseline
  - 이번 단계의 Baseline을 아래의 링크에서 확인할 수 있습니다.

  URL: https://tvm.apache.org/docs/tutorials/optimize/opt_conv_cuda.html

[Step 3] Optimization Tensor Operators

- Optimize Convolution on GPU
  - Preparation and Algorithm

```
import numpy as np
import tvm
from tvm import te

# The sizes of inputs and filters
batch = 256
in_channel = 256
out_channel = 512
in_size = 14
kernel = 3
pad = 1
stride = 1

# Algorithm
A = te.placeholder((in_size, in_size, in_channel, batch), name="A")
W = te.placeholder((kernel, kernel, in_channel, out_channel), name="W")
out_size = (in_size - kernel + 2 * pad) // stride + 1
# Pad input
Apad = te.compute(
    (in_size + 2 * pad, in_size + 2 * pad, in_channel, batch),
    lambda yy, xx, cc, nn: tvm.tir.if_then_else(
        tvm.tir.all(yy >= pad, yy - pad < in_size, xx >= pad, xx - pad < in_size),
        A[yy - pad, xx - pad, cc, nn],
        tvm.tir.const(0.0, "float32"),
    ),
    name="Apad",
)
# Create reduction variables
rc = te.reduce_axis((0, in_channel), name="rc")
ry = te.reduce_axis((0, kernel), name="ry")
rx = te.reduce_axis((0, kernel), name="rx")
# Compute the convolution
B = te.compute(
    (out_size, out_size, out_channel, batch),
    lambda yy, xx, ff, nn: te.sum(
        Apad[yy * stride + ry, xx * stride + rx, rc, nn] * W[ry, rx, rc, ff], axis=[ry, rx, rc]
    ),
    name="B",
)
```

[Step 3] Optimization Tensor Operators

- Optimize Convolution on GPU
  - Memory Hierarchy and Blocking

```
# Memory Hierarchy
# Designate the memory hierarchy
s = te.create_schedule(B.op)
s[Apad].compute_inline()  # compute Apad inline
AA = s.cache_read(Apad, "shared", [B])
WW = s.cache_read(W, "shared", [B])
AL = s.cache_read(AA, "local", [B])
WL = s.cache_read(WW, "local", [B])
BL = s.cache_write(B, "local")

# Blocking
# tile consts
tile = 8
num_thread = 8
block_factor = tile * num_thread
step = 8
vthread = 2

# Get the GPU thread indices:q
block_x = te.thread_axis("blockIdx.x")
block_y = te.thread_axis("blockIdx.y")
block_z = te.thread_axis("blockIdx.z")
thread_x = te.thread_axis((0, num_thread), "threadIdx.x")
thread_y = te.thread_axis((0, num_thread), "threadIdx.y")
thread_xz = te.thread_axis((0, vthread), "vthread", name="vx")
thread_yz = te.thread_axis((0, vthread), "vthread", name="vy")

# Split the workloads
hi, wi, fi, ni = s[B].op.axis
bz = s[B].fuse(hi, wi)
by, fi = s[B].split(fi, factor=block_factor)
bx, ni = s[B].split(ni, factor=block_factor)

# Bind the iteration variables to GPU thread indices
s[B].bind(bz, block_z)
s[B].bind(by, block_y)
s[B].bind(bx, block_x)
```


Hanyang University
Division of Computer Science & Engineering

[Step 3] Optimization Tensor Operators

- Optimize Convolution on GPU
  - Virtual thread Split and Cooperative Fetching

```
# Virtual Thread Split
tyz, fi = s[B].split(fi, nparts=vthread)  # virtual thread split
txz, ni = s[B].split(ni, nparts=vthread)  # virtual thread split
ty, fi = s[B].split(fi, nparts=num_thread)
tx, ni = s[B].split(ni, nparts=num_thread)
s[B].reorder(bz, by, bx, tyz, txz, ty, tx, fi, ni)

s[B].bind(tyz, thread_yz)
s[B].bind(txz, thread_xz)
s[B].bind(ty, thread_y)
s[B].bind(tx, thread_x)

# Cooperative Fetching
# Schedule BL local write
s[BL].compute_at(s[B], tx)
yi, xi, fi, ni = s[BL].op.axis
ry, rx, rc = s[BL].op.reduce_axis
rco, rci = s[BL].split(rc, factor=step)
s[BL].reorder(rco, ry, rx, rci, fi, ni)

# Attach computation to iteration variables
s[AA].compute_at(s[BL], rx)
s[WW].compute_at(s[BL], rx)
s[AL].compute_at(s[BL], rci)
s[WL].compute_at(s[BL], rci)

# Schedule for A's shared memory load
yi, xi, ci, ni = s[AA].op.axis
ty, ci = s[AA].split(ci, nparts=num_thread)
tx, ni = s[AA].split(ni, nparts=num_thread)
_, ni = s[AA].split(ni, factor=4)
s[AA].reorder(ty, tx, yi, xi, ci, ni)
s[AA].bind(ty, thread_y)
s[AA].bind(tx, thread_x)
s[AA].vectorize(ni)  # vectorize memory load

# Schedule for W's shared memory load
yi, xi, ci, fi = s[WW].op.axis
ty, ci = s[WW].split(ci, nparts=num_thread)
tx, fi = s[WW].split(fi, nparts=num_thread)
_, fi = s[WW].split(fi, factor=4)
s[WW].reorder(ty, tx, yi, xi, ci, fi)
s[WW].bind(ty, thread_y)
s[WW].bind(tx, thread_x)
s[WW].vectorize(fi)  # vectorize memory load
```

[Step 3] Optimization Tensor Operators

- Optimize Convolution on GPU
  - Generate CUDA Kernel
  - 실행결과 출력 부분

```
# Generate CUDA Kernel
func = tvm.build(s, [A, W, B], "cuda")
dev = tvm.cuda(0)
a_np = np.random.uniform(size=(in_size, in_size, in_channel, batch)).astype(A.dtype)
w_np = np.random.uniform(size=(kernel, kernel, in_channel, out_channel)).astype(W.dtype)
a = tvm.nd.array(a_np, dev)
w = tvm.nd.array(w_np, dev)
b = tvm.nd.array(np.zeros((out_size, out_size, out_channel, batch), dtype=B.dtype), dev)
func(a, w, b)
evaluator = func.time_evaluator(func.entry_name, dev, number=1)
print("Convolution: %f ms" % (evaluator(a, w, b).mean * 1e3))
```

[Step 3] Optimization Tensor Operators

- Optimize Convolution on GPU
  - 아래와 같은 출력결과를 Screenshot으로 제출

```
Convolution: 78.342357 ms
```

[Step 4] Relay Pass Infra

- Relay Pass Infra
  - 해당 단계는 생성한 Relay 프로그램을 최적화 Pass 들을 사용해서 초기 프로그램과 비교하는 과정을 수행해본다.
  - 해당 단계에서는 최적화에 사용된 Constant Fold, Eliminate Common subexpression, Optimized Fuse 옵션을 확인하고, 사용하기 전후를 비교한다.
- Baseline
  - 이번 예제의 Baseline을 아래의 링크에서 확인할 수 있습니다.

  URL: https://tvm.apache.org/docs/tutorials/dev/use_pass_infra.html

[Step 4] Relay Pass Infra

- Relay Pass Infra
  - relay_pass_infra.py

```python
import numpy as np
import tvm
from tvm import te
import tvm.relay as relay

# Create An Example Relay Program
def example():
    shape = (1, 64, 54, 54)
    c_data = np.empty(shape).astype("float32")
    c = relay.const(c_data)
    weight = relay.var("weight", shape=(64, 64, 3, 3))
    x = relay.var("x", relay.TensorType((1, 64, 56, 56), "float32"))
    conv = relay.nn.conv2d(x, weight)
    y = relay.add(c, c)
    y = relay.multiply(y, relay.const(2, "float32"))
    y = relay.add(conv, y)
    z = relay.add(y, c)
    z1 = relay.add(y, c)
    z2 = relay.add(z, z1)
    return relay.Function([x, weight], z2)

# Optimize the Program

f = example()
mod = tvm.IRModule.from_expr(f)
print(mod)

fold_const = relay.transform.FoldConstant()
mod = fold_const(mod)
mod = relay.transform.EliminateCommonSubexpr()(mod)
mod = relay.transform.FuseOps(fuse_opt_level=2)(mod)
print(mod)

print("done")
```

[Step 4] Relay Pass Infra

- Relay Pass Infra
  - 실행 결과 출력 부분

```python
import numpy as np
import tvm
from tvm import te
import tvm.relay as relay

# Create An Example Relay Program
def example():
    shape = (1, 64, 54, 54)
    c_data = np.empty(shape).astype("float32")
    c = relay.const(c_data)
    weight = relay.var("weight", shape=(64, 64, 3, 3))
    x = relay.var("x", relay.TensorType((1, 64, 56, 56), "float32"))
    conv = relay.nn.conv2d(x, weight)
    y = relay.add(c, c)
    y = relay.multiply(y, relay.const(2, "float32"))
    y = relay.add(conv, y)
    z = relay.add(y, c)
    z1 = relay.add(y, c)
    z2 = relay.add(z, z1)
    return relay.Function([x, weight], z2)

# Optimize the Program

f = example()
mod = tvm.IRModule.from_expr(f)
print(mod)

fold_const = relay.transform.FoldConstant()
mod = fold_const(mod)
mod = relay.transform.EliminateCommonSubexpr()(mod)
mod = relay.transform.FuseOps(fuse_opt_level=2)(mod)
print(mod)

print("done")
```

[Step 4] Relay Pass Infra

- Relay Pass Infra
  - 아래의 실행 결과를 screenshot으로 제출
  - 반드시 마지막에 Done 문구가 출력되어야 함.

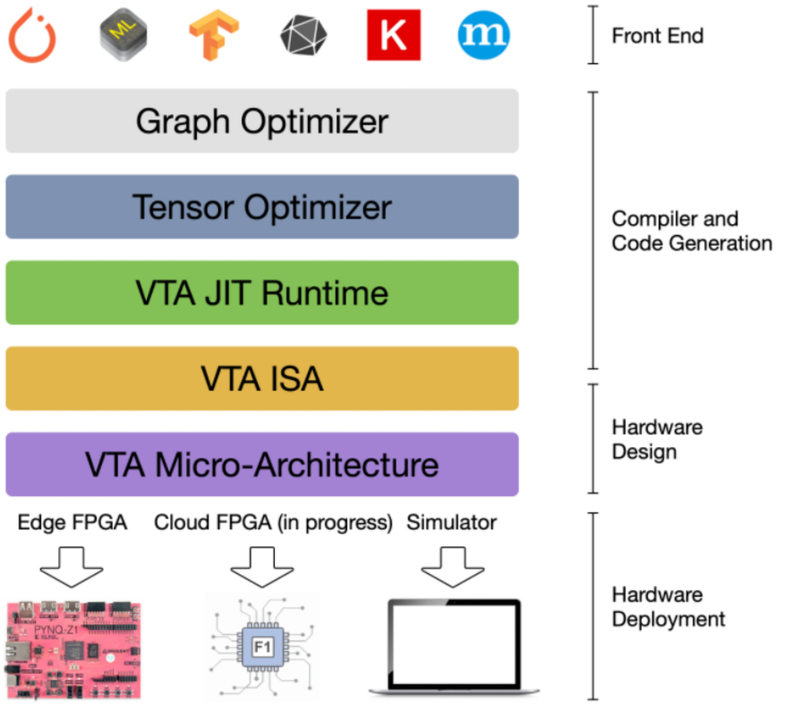
```
def @main(%x: Tensor[(1, 64, 56, 56), float32], %weight: Tensor[(64, 64, 3, 3), float32]) {
  %0 = add(meta[relay.Constant][0], meta[relay.Constant][0]);
  %1 = nn.conv2d(%x, %weight, padding=[0, 0, 0, 0]);
  %2 = multiply(%0, 2f);
  %3 = add(%1, %2);
  %4 = add(%3, meta[relay.Constant][0]);
  %5 = add(%3, meta[relay.Constant][0]);
  add(%4, %5)
}


def @main(%x: Tensor[(1, 64, 56, 56), float32], %weight: Tensor[(64, 64, 3, 3), float32]) -> Tensor[(1, 64, 54, 54), float32] {
  %3 = fn (%p0: Tensor[(1, 64, 56, 56), float32], %p1: Tensor[(64, 64, 3, 3), float32], %p2: Tensor[(1, 64, 54, 54), float32], %p3: Tensor[(1, 64, 54, 54), float32], Primitive=1) -> Tensor[(1, 64, 54, 54), float32] {
    %0 = nn.conv2d(%p0, %p1, padding=[0, 0, 0, 0]) /* ty=Tensor[(1, 64, 54, 54), float32] */;
    %1 = add(%0, %p2) /* ty=Tensor[(1, 64, 54, 54), float32] */;
    %2 = add(%1, %p3) /* ty=Tensor[(1, 64, 54, 54), float32] */;
    add(%2, %2) /* ty=Tensor[(1, 64, 54, 54), float32] */
  };
  %3(%x, %weight, meta[relay.Constant][0] /* ty=Tensor[(1, 64, 54, 54), float32] */, meta[relay.Constant][1] /* ty=Tensor[(1, 64, 54, 54), float32] */) /* ty=Tensor[(1, 64, 54, 54), float32] */
}


done
```

VTA

- VTA (Versatile Tensor Accelerator)
  - 딥러닝 가속기 설계를 위한 확장된 TVM 프레임워크 stack

- VTA의 특징
  - Simulator를 통해 FPGA가 없어도 테스팅이 가능하다.
  - RPC 를 사용하여 원격지의 Edge 및 Cloud 환경에서도 사용 가능
  - Python 스크립트를 통해, 미리 만든 Bitstream을 FPGA에 쉽게 적용 가능
  - Open Source 이기 때문에 원하는 대로 변경 가능

- VTA의 한계
  - Open Source 이기 때문에, 지속적인 변경이 있음.
  - Training 이 불가능.



[Step 5] VTA Simulation

- VTA Simulation
  - 이번 단계에서는 VTA Simulator를 사용하여, Vector Add 연산 테스팅을 수행한다.
  - 이번 단계에서는 VTA Simulator 환경에서, DMA Transfer와 ALU Operation을 활용하고, build된 모듈을 사용하여 VTA의 save, Load, run, verify의 과정을 수행한다.
- Pre-requisite
  - 본 과제는 보드가 아닌 Simulation을 수행하기 때문에, TVM 빌드 시, 반드시 config.cmake 파일을 아래와 같이 변경하고 빌드해야 한다.

```
# Whether to build fast VTA simulator driver
set(USE_VTA_FSIM ON)
```

  - 환경 변수를 아래와 같이 설정 해야 한다.

RPC 주소 및 포트 설정 (같은 주소와 포트로 설정할 필요 없음), VTA를 위한 library 경로 설정

```
export VTA_USER_HOME=~/tvm-0.8.dev0/vta
export PYTHONPATH=$VTA_USER_HOME/python:${PYTHONPATH}

export VTA_RPC_HOST="192.168.0.9"
export VTA_RPC_PORT=9091
```

[Step 5] VTA Simulation

- Baseline
  - 이번 단계의 Baseline을 아래의 링크에서 확인할 수 있습니다.

  URL: https://tvm.apache.org/docs/vta/tutorials/vta_get_started.html

[Step 5] VTA Simulation

- VTA Simulation
  - vta_get_started.py

VTA 초기 설정

```python
from __future__ import absolute_import, print_function

import os
import tvm
from tvm import te
import vta
import numpy as np

# Loading in VTA Parameters
env = vta.get_env()

# FPGA Programming
from tvm import rpc
from tvm.contrib import utils
from vta.testing import simulator

# We read the Pynq RPC host IP address and port number from the OS environment
host = os.environ.get("VTA_RPC_HOST", "192.168.2.99")
port = int(os.environ.get("VTA_RPC_PORT", "9091"))

if env.TARGET == "pynq" or env.TARGET == "de10nano":

    # Make sure that TVM was compiled with RPC=1
    assert tvm.runtime.enabled("rpc")
    remote = rpc.connect(host, port)

    # Reconfigure the JIT runtime
    vta.reconfig_runtime(remote)
    vta.program_fpga(remote, bitstream=None)

# In simulation mode, host the RPC server locally.
elif env.TARGET in ("sim", "tsim", "intelfocl"):
    remote = rpc.LocalSession()
    if env.TARGET in ["intelfocl"]:
        # program intelfocl aocx
        vta.program_fpga(remote, bitstream="vta.bitstream")
```

연산 Scheduling

```python
m = 64
o = 1
# A placeholder tensor in tiled data format
A = te.placeholder((o, m, env.BATCH, env.BLOCK_OUT), name="A", dtype=env.acc_dtype)
# B placeholder tensor in tiled data format
B = te.placeholder((o, m, env.BATCH, env.BLOCK_OUT), name="B", dtype=env.acc_dtype)

# Copy Buffers
# A copy buffer
A_buf = te.compute((o, m, env.BATCH, env.BLOCK_OUT), lambda *i: A(*i), "A_buf")
# B copy buffer
B_buf = te.compute((o, m, env.BATCH, env.BLOCK_OUT), lambda *i: B(*i), "B_buf")

# Vector Addition
# Describe the in-VTA vector addition
C_buf = te.compute(
    (o, m, env.BATCH, env.BLOCK_OUT),
    lambda *i: A_buf(*i).astype(env.acc_dtype) + B_buf(*i).astype(env.acc_dtype),
    name="C_buf")

# Casting the Results
# Cast to output type, and send to main memory
C = te.compute(
    (o, m, env.BATCH, env.BLOCK_OUT), lambda *i: C_buf(*i).astype(env.inp_dtype), name="C")

# Default Schedule
# Let's take a look at the generated schedule
s = te.create_schedule(C.op)

# Buffer Scopes
# Set the intermediate tensors' scope to VTA's on-chip accumulator buffer
s[A_buf].set_scope(env.acc_scope)
s[B_buf].set_scope(env.acc_scope)
s[C_buf].set_scope(env.acc_scope)

# DMA Transfers
# Tag the buffer copies with the DMA pragma to map a copy loop to a DMA transfer operation
s[A_buf].pragma(s[A_buf].op.axis[0], env.dma_copy)
s[B_buf].pragma(s[B_buf].op.axis[0], env.dma_copy)
s[C].pragma(s[C].op.axis[0], env.dma_copy)

# ALU Operations
# Tell TVM that the computation needs to be performed on VTA's vector ALU
s[C_buf].pragma(C_buf.op.axis[0], env.alu)

print(vta.lower(s, [A, B, C], simple_mode=True))
```


Hanyang University
Division of Computer Science & Engineering

[Step 5] VTA Simulation

- VTA Simulation
  - vta_get_started.py

VTA 빌드, 실행 및 검증

```
# TVM Compilation
my_vadd = vta.build(s, [A, B, C], "ext_dev", env.target_host, name="my_vadd")

# Get the remote device context
ctx = remote.ext_dev(0)

# Initialize the A and B arrays randomly in the int range of (-128, 128]
A_orig = np.random.randint(-128, 128, size=(o * env.BATCH, m * env.BLOCK_OUT)).astype(A.dtype)
B_orig = np.random.randint(-128, 128, size=(o * env.BATCH, m * env.BLOCK_OUT)).astype(B.dtype)

# Apply packing to the A and B arrays from a 2D to a 4D packed layout
A_packed = A_orig.reshape(o, env.BATCH, m, env.BLOCK_OUT).transpose((0, 2, 1, 3))
B_packed = B_orig.reshape(o, env.BATCH, m, env.BLOCK_OUT).transpose((0, 2, 1, 3))

# Format the input/output arrays with tvm.nd.array to the DLPack standard
A_nd = tvm.nd.array(A_packed, ctx)
B_nd = tvm.nd.array(B_packed, ctx)
C_nd = tvm.nd.array(np.zeros((o, m, env.BATCH, env.BLOCK_OUT)).astype(C.dtype), ctx)

temp = utils.tempdir()
my_vadd.save(temp.relpath("vadd.o"))
remote.upload(temp.relpath("vadd.o"))
f = remote.load_module("vadd.o")

# Invoke the module to perform the computation
f(A_nd, B_nd, C_nd)

# Verifying Correctness
# Compute reference result with numpy
C_ref = (A_orig.astype(env.acc_dtype) + B_orig.astype(env.acc_dtype)).astype(C.dtype)
C_ref = C_ref.reshape(o, env.BATCH, m, env.BLOCK_OUT).transpose((0, 2, 1, 3))
np.testing.assert_equal(C_ref, C_nd.numpy())
print("Successful vector add test!")
```

[Step 5] VTA Simulation

- VTA Simulation
  - 반드시 아래의 Successful vector add test! 문구가 나와야 함.
  - **아래의 실행 결과를 screenshot으로 제출**

```
primfn(A_1: handle, B_1: handle, C_1: handle) -> ()
  attr = {"global_symbol": "main", "tir.noalias": True}
  buffers = {C: Buffer(C_2: Pointer(int8), int8, [1, 64, 1, 16], []),
             A: Buffer(A_2: Pointer(int32), int32, [1, 64, 1, 16], []),
             B: Buffer(B_2: Pointer(int32), int32, [1, 64, 1, 16], [])}
  buffer_map = {A_1: A, B_1: B, C_1: C} {
  attr [A_buf: Pointer(int32)] "storage_scope" = "local.acc_buffer" {
    attr [IterVar(vta: int32, (nullptr), "ThreadIndex", "vta")] "coproc_scope" = 2 {
      @tir.call_extern("VTALoadBuffer2D", @tir.tvm_thread_context(@tir.vta.command_handle(, dtype=handle), dtype=handle), A_2, 0, 64, 1, 64, 0, 0, 0, 0, 0, 3, dtype=int32)
      @tir.call_extern("VTALoadBuffer2D", @tir.tvm_thread_context(@tir.vta.command_handle(, dtype=handle), dtype=handle), B_2, 0, 64, 1, 64, 0, 0, 0, 0, 64, 3, dtype=int32)
      attr [IterVar(vta, (nullptr), "ThreadIndex", "vta")] "coproc_uop_scope" = "VTAPushALUOp" {
        @tir.call_extern("VTAUopLoopBegin", 64, 1, 1, 0, dtype=int32)
        @tir.vta.uop_push(1, 0, 0, 64, 0, 2, 0, 0, dtype=int32)
        @tir.call_extern("VTAUopLoopEnd", dtype=int32)
      }
      @tir.vta.coproc_dep_push(2, 3, dtype=int32)
    }
    attr [IterVar(vta, (nullptr), "ThreadIndex", "vta")] "coproc_scope" = 3 {
      @tir.vta.coproc_dep_pop(2, 3, dtype=int32)
      @tir.call_extern("VTAStoreBuffer2D", @tir.tvm_thread_context(@tir.vta.command_handle(, dtype=handle), dtype=handle), 0, 4, C_2, 0, 64, 1, 64, dtype=int32)
    }
    @tir.vta.coproc_sync(, dtype=int32)
  }
}


Successful vector add test!
```

Thanks