Shuffle 层

- 初始示例代码
- first_transpose
- reshape_dims
- second_transpose
- 组合使用的例子
- zero_is_placeholder
- set_input
  - 静态 set_input
  - 动态 set_input（使用 context.set_shape_input）
  - dynamic shape 模式下的 shuffle + set_input（使用 context.set_binding_shape）

初始示例代码

```
import numpy as np
from cuda import cudart
import tensorrt as trt

nIn, cIn, hIn, wIn = 1, 3, 4, 5  # 输入张量 NCHW
data = np.arange(cIn, dtype=np.float32).reshape(cIn, 1, 1) * 100 + np.arange(hIn).reshape(1, hIn, 1) *
10 + np.arange(wIn).reshape(1, 1, wIn)  # 输入数据
data = data.reshape(nIn, cIn, hIn, wIn).astype(np.float32)

np.set_printoptions(precision=8, linewidth=200, suppress=True)
cudart.cudaDeviceSynchronize()

logger = trt.Logger(trt.Logger.ERROR)
builder = trt.Builder(logger)
network = builder.create_network(1 << int(trt.NetworkDefinitionCreationFlag.EXPLICIT_BATCH))
config = builder.create_builder_config()
config.max_workspace_size = 1 << 30
inputT0 = network.add_input('inputT0', trt.DataType.FLOAT, (nIn, cIn, hIn, wIn))
#---------------------------------------------------------- --------------------# 替换部分
shuffleLayer = network.add_shuffle(inputT0)
#---------------------------------------------------------- --------------------# 替换部分
network.mark_output(shuffleLayer.get_output(0))
engineString = builder.build_serialized_network(network, config)
engine = trt.Runtime(logger).deserialize_cuda_engine(engineString)
context = engine.create_execution_context()
_, stream = cudart.cudaStreamCreate()

inputH0 = np.ascontiguousarray(data.reshape(-1))
outputH0 = np.empty(context.get_binding_shape(1), dtype=trt.nptype(engine.get_binding_dtype(1)))
_, inputD0 = cudart.cudaMallocAsync(inputH0.nbytes, stream)
_, outputD0 = cudart.cudaMallocAsync(outputH0.nbytes, stream)

cudart.cudaMemcpyAsync(inputD0, inputH0.ctypes.data, inputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
context.execute_async_v2([int(inputD0), int(outputD0)], stream)
```

```
cudart.cudaMemcpyAsync(outputH0.ctypes.data, outputD0, outputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
cudart.cudaStreamSynchronize(stream)

print("inputH0 :", data.shape)
print(data)
print("outputH0:", outputH0.shape)
print(outputH0)

cudart.cudaStreamDestroy(stream)
cudart.cudaFree(inputD0)
cudart.cudaFree(outputD0)
```

- 输入张量形状 (1,3,4,5)，百位、十位、个位分别表示 CHW 维编号

$$\left[\left[\begin{bmatrix}0. & 1. & 2. & 3. & 4.\\ 10. & 11. & 12. & 13. & 14.\\ 20. & 21. & 22. & 23. & 24.\\ 30. & 31. & 32. & 33. & 34.\end{bmatrix}\begin{bmatrix}100. & 101. & 102. & 103. & 104.\\ 110. & 111. & 112. & 113. & 114.\\ 120. & 121. & 122. & 123. & 124.\\ 130. & 131. & 132. & 133. & 134.\end{bmatrix}\begin{bmatrix}200. & 201. & 202. & 203. & 204.\\ 210. & 211. & 212. & 213. & 214.\\ 220. & 221. & 222. & 223. & 224.\\ 230. & 231. & 232. & 233. & 234.\end{bmatrix}\right]\right]$$

- 不指定 shuffle 参数的情况下，输出张量形状 (1,3,4,5)，张量不发生变化

$$\left[\left[\begin{bmatrix}0. & 1. & 2. & 3. & 4.\\ 10. & 11. & 12. & 13. & 14.\\ 20. & 21. & 22. & 23. & 24.\\ 30. & 31. & 32. & 33. & 34.\end{bmatrix}\begin{bmatrix}100. & 101. & 102. & 103. & 104.\\ 110. & 111. & 112. & 113. & 114.\\ 120. & 121. & 122. & 123. & 124.\\ 130. & 131. & 132. & 133. & 134.\end{bmatrix}\begin{bmatrix}200. & 201. & 202. & 203. & 204.\\ 210. & 211. & 212. & 213. & 214.\\ 220. & 221. & 222. & 223. & 224.\\ 230. & 231. & 232. & 233. & 234.\end{bmatrix}\right]\right]$$

first_transpose

```
shuffleLayer = network.add_shuffle(inputT0)
shuffleLayer.first_transpose = (0, 2, 1, 3)  # 首次转置，默认值 (0,1,2,...)
```

- 指定 first_transpose=(0,2,1，3)，输出张量形状 (1,4,3,5)，将第 0、1、2、3 维（原始顺序）分别放置到第 0、2、1、3 维（指定顺序）的位置

$$\left[\left[\begin{bmatrix}0. & 1. & 2. & 3. & 4.\\ 100. & 101. & 102. & 103. & 104.\\ 200. & 201. & 202. & 203. & 204.\end{bmatrix}\begin{bmatrix}10. & 11. & 12. & 13. & 14.\\ 110. & 111. & 112. & 113. & 114.\\ 210. & 211. & 212. & 213. & 214.\end{bmatrix}\begin{bmatrix}20. & 21. & 22. & 23. & 24.\\ 120. & 121. & 122. & 123. & 124.\\ 220. & 221. & 222. & 223. & 224.\end{bmatrix}\begin{bmatrix}30. & 31. & 32. & 33. & 34.\\ 130. & 131. & 132. & 133. & 134.\\ 230. & 231. & 232. & 233. & 234.\end{bmatrix}\right]\right]$$

reshape_dims

```
shuffleLayer = network.add_shuffle(inputT0)
shuffleLayer.reshape_dims = (-1, 2, 15)  # 指定新形状，至多有一个 -1 表示自动计算，默认值 inputT0.shape
```

- 指定 reshape_dims=(-1,2,15)，输出张量形状 (2,2,15)，保持原来元素顺序的条件下调整张量形状。可以使用至多一个 -1 自动计算

$$\left[\begin{bmatrix}0. & 1. & 2. & 3. & 4. & 10. & 11. & 12. & 13. & 14. & 20. & 21. & 22. & 23. & 24.\\ 30. & 31. & 32. & 33. & 34. & 100. & 101. & 102. & 103. & 104. & 110. & 111. & 112. & 113. & 114.\end{bmatrix}\right.$$
$$\left.\begin{bmatrix}120. & 121. & 122. & 123. & 124. & 130. & 131. & 132. & 133. & 134. & 200. & 201. & 202. & 203. & 204.\\ 210. & 211. & 212. & 213. & 214. & 220. & 221. & 222. & 223. & 224. & 230. & 231. & 232. & 233. & 234.\end{bmatrix}\right]$$

```
shuffleLayer = network.add_shuffle(inputT0)
shuffleLayer.reshape_dims = (0, 0, -1)
```

- 指定 reshape_dims=(0,0,-1)，输出张量形状 (1,3,20)，0 表示照搬输入张量形状的相应位置上的值，这里两个 0 保持了输入张量形状的最高两维 1 和 3，剩下的自动计算

```
[[[ 0.  1.  2.  3.  4. 10. 11. 12. 13. 14. 20. 21. 22. 23. 24. 30. 31. 32. 33. 34.]
  [100. 101. 102. 103. 104. 110. 111. 112. 113. 114. 120. 121. 122. 123. 124. 130. 131. 132. 133. 134.]
  [200. 201. 202. 203. 204. 210. 211. 212. 213. 214. 220. 221. 222. 223. 224. 230. 231. 232. 233. 234.]]]
```

---

second_transpose

```
shuffleLayer = network.add_shuffle(inputT0)
shuffleLayer.second_transpose = (0, 2, 1, 3)  # 末次转置，默认值 inputT0.reshape(...).shape
```

- 指定 second_transpose=(0,2,1,3)，输出张量形状 (1,4,3,5)，单独使用时结果与 first_transpose 示例相同，但是发生在调整形状之后

组合使用的例子

```
shuffleLayer = network.add_shuffle(inputT0)
shuffleLayer.first_transpose = (0, 2, 1, 3)
shuffleLayer.reshape_dims = (1, 4, 5, 3)
shuffleLayer.second_transpose = (0, 2, 1, 3)
```

- 输出张量形状 (1,5,4,3)

$$\left[\left[\begin{bmatrix}0. & 1. & 2.\\ 10. & 11. & 12.\\ 20. & 21. & 22.\\ 30. & 31. & 32.\end{bmatrix}\begin{bmatrix}3. & 4. & 100.\\ 13. & 14. & 110.\\ 23. & 24. & 120.\\ 33. & 34. & 130.\end{bmatrix}\begin{bmatrix}101. & 102. & 103.\\ 111. & 112. & 113.\\ 121. & 122. & 123.\\ 131. & 132. & 133.\end{bmatrix}\begin{bmatrix}104. & 200. & 201.\\ 114. & 210. & 211.\\ 124. & 220. & 221.\\ 134. & 230. & 231.\end{bmatrix}\begin{bmatrix}202. & 203. & 204.\\ 212. & 213. & 214.\\ 222. & 223. & 224.\\ 232. & 233. & 234.\end{bmatrix}\right]\right]$$

- 计算过程：

$$\left[\left[\begin{bmatrix}0. & 1. & 2. & 3. & 4.\\ 10. & 11. & 12. & 13. & 14.\\ 20. & 21. & 22. & 23. & 24.\\ 30. & 31. & 32. & 33. & 34.\end{bmatrix}\begin{bmatrix}100. & 101. & 102. & 103. & 104.\\ 110. & 111. & 112. & 113. & 114.\\ 120. & 121. & 122. & 123. & 124.\\ 130. & 131. & 132. & 133. & 134.\end{bmatrix}\begin{bmatrix}200. & 201. & 202. & 203. & 204.\\ 210. & 211. & 212. & 213. & 214.\\ 220. & 221. & 222. & 223. & 224.\\ 230. & 231. & 232. & 233. & 234.\end{bmatrix}\right]\right]$$

$$\Downarrow first\_transpose\,(0,2,1,3)$$

$$\left[\left[\begin{bmatrix}0. & 1. & 2. & 3. & 4.\\ 100. & 101. & 102. & 103. & 104.\\ 200. & 201. & 202. & 203. & 204.\end{bmatrix}\begin{bmatrix}10. & 11. & 12. & 13. & 14.\\ 110. & 111. & 112. & 113. & 114.\\ 210. & 211. & 212. & 213. & 214.\end{bmatrix}\begin{bmatrix}20. & 21. & 22. & 23. & 24.\\ 120. & 121. & 122. & 123. & 124.\\ 220. & 221. & 222. & 223. & 224.\end{bmatrix}\begin{bmatrix}30. & 31. & 32. & 33. & 34.\\ 130. & 131. & 132. & 133. & 134.\\ 230. & 231. & 232. & 233. & 234.\end{bmatrix}\right]\right]$$

$$\Downarrow reshape\,(1,4,5,3)$$

$$\left[\left[\begin{bmatrix}0. & 1. & 2.\\ 3. & 4. & 100.\\ 101. & 102. & 103.\\ 104. & 200. & 201.\\ 202. & 203. & 204.\end{bmatrix}\begin{bmatrix}10. & 11. & 12.\\ 13. & 14. & 110.\\ 111. & 112. & 113.\\ 114. & 210. & 211.\\ 212. & 213. & 214.\end{bmatrix}\begin{bmatrix}20. & 21. & 22.\\ 23. & 24. & 120.\\ 121. & 122. & 123.\\ 124. & 220. & 221.\\ 222. & 223. & 224.\end{bmatrix}\begin{bmatrix}30. & 31. & 32.\\ 33. & 34. & 130.\\ 131. & 132. & 133.\\ 134. & 230. & 231.\\ 232. & 233. & 234.\end{bmatrix}\right]\right]$$

$$\Downarrow second\_transpose\,(0,2,1,3)$$

$$\left[\left[\begin{bmatrix}0. & 1. & 2.\\ 10. & 11. & 12.\\ 20. & 21. & 22.\\ 30. & 31. & 32.\end{bmatrix}\begin{bmatrix}3. & 4. & 100.\\ 13. & 14. & 110.\\ 23. & 24. & 120.\\ 33. & 34. & 130.\end{bmatrix}\begin{bmatrix}101. & 102. & 103.\\ 111. & 112. & 113.\\ 121. & 122. & 123.\\ 131. & 132. & 133.\end{bmatrix}\begin{bmatrix}104. & 200. & 201.\\ 114. & 210. & 211.\\ 124. & 220. & 221.\\ 134. & 230. & 231.\end{bmatrix}\begin{bmatrix}202. & 203. & 204.\\ 212. & 213. & 214.\\ 222. & 223. & 224.\\ 232. & 233. & 234.\end{bmatrix}\right]\right]$$

---

zero_is_placeholder

```
shuffleLayer = network.add_shuffle(inputT0)
shuffleLayer.zero_is_placeholder = True  # 使用 0 模式，默认值 True
shuffleLayer.reshape_dims = (0, 0, 0, 0)
```

- 输出张量形状 (1,3,4,5)，结果与初始示例代码相同，0 表示照搬输入张量形状的相应位置上的值

$$\left[\left[\begin{matrix}0. & 1. & 2. & 3. & 4.\\ 10. & 11. & 12. & 13. & 14.\\ 20. & 21. & 22. & 23. & 24.\\ 30. & 31. & 32. & 33. & 34.\end{matrix}\right]\left[\begin{matrix}100. & 101. & 102. & 103. & 104.\\ 110. & 111. & 112. & 113. & 114.\\ 120. & 121. & 122. & 123. & 124.\\ 130. & 131. & 132. & 133. & 134.\end{matrix}\right]\left[\begin{matrix}200. & 201. & 202. & 203. & 204.\\ 210. & 211. & 212. & 213. & 214.\\ 220. & 221. & 222. & 223. & 224.\\ 230. & 231. & 232. & 233. & 234.\end{matrix}\right]\right]$$

```
constantLayer = network.add_constant([0], trt.Weights(trt.DataType.FLOAT))  # 静态空层
shuffleLayer = network.add_shuffle(constantLayer.get_output(0))
shuffleLayer.zero_is_placeholder = False
shuffleLayer.reshape_dims = (1, 3, 4, 0)  # 对齐另一个张量的形状
concatenationLayer = network.add_concatenation([inputT0, shuffleLayer.get_output(0)])
concatenationLayer.axis = 3
```

- 输出张量形状 (1,3,4,5)（注意输出张量改成 concatenationLayer.get_output(0)），结果与初始示例代码相同
- 这种用法常用于本层输出张量广播后再用于其他层的情况，参见 09-Advance 的“EmptyTensor”部分

---

set_input

静态 set_input

```
constantLayer = network.add_constant([4], np.array([1, 4, 5, 3], dtype=np.int32))  # 静态新形状
shuffleLayer = network.add_shuffle(inputT0)
#shuffleLayer.set_input(0,inputT0)                                                  # 0
号输入是被 shuffle 的张量
shuffleLayer.set_input(1, constantLayer.get_output(0))  # 1 号输入是新形状张量
```

- 输出张量形状 (1,4,5,3)

$$\left[\left[\left[\begin{matrix}0. & 1. & 2.\\ 3. & 4. & 10.\\ 11. & 12. & 13.\\ 14. & 20. & 21.\\ 22. & 23. & 24.\end{matrix}\right]\left[\begin{matrix}30. & 31. & 32.\\ 33. & 34. & 100.\\ 101. & 102. & 103.\\ 104. & 110. & 111.\\ 112. & 113. & 114.\end{matrix}\right]\left[\begin{matrix}120. & 121. & 122.\\ 123. & 124. & 130.\\ 131. & 132. & 133.\\ 134. & 200. & 201.\\ 202. & 203. & 204.\end{matrix}\right]\left[\begin{matrix}210. & 211. & 212.\\ 213. & 214. & 220.\\ 221. & 222. & 223.\\ 224. & 230. & 231.\\ 232. & 233. & 234.\end{matrix}\right]\right]\right]$$

动态 set_input（使用 context.set_shape_input）

```
import numpy as np
from cuda import cudart
import tensorrt as trt

nIn, cIn, hIn, wIn = 1, 3, 4, 5
data0 = np.arange(cIn, dtype=np.float32).reshape(cIn, 1, 1) * 100 + np.arange(hIn).reshape(1, hIn, 1) *
10 + np.arange(wIn).reshape(1, 1, wIn)
data0 = data0.reshape(nIn, cIn, hIn, wIn).astype(np.float32)
data1 = np.array([1, 4, 5, 3], dtype=np.int32)

np.set_printoptions(precision=8, linewidth=200, suppress=True)
cudart.cudaDeviceSynchronize()

logger = trt.Logger(trt.Logger.ERROR)
builder = trt.Builder(logger)
network = builder.create_network(1 << int(trt.NetworkDefinitionCreationFlag.EXPLICIT_BATCH))
profile = builder.create_optimization_profile()  # 需要使用 profile
config = builder.create_builder_config()
config.max_workspace_size = 1 << 30
inputT0 = network.add_input('inputT0', trt.DataType.FLOAT, (nIn, cIn, hIn, wIn))
inputT1 = network.add_input('inputT1', trt.DataType.INT32, (4, ))
```

```
profile.set_shape_input(inputT1.name, (1, 1, 1, 1), (nIn, cIn, hIn, wIn), (5, 5, 5, 5))  # 这里设置的不是 shape input 的形状而是值，范围覆盖住之后需要的值就好
config.add_optimization_profile(profile)

shuffleLayer = network.add_shuffle(inputT0)
#shuffleLayer.set_input(0,inputT0)
shuffleLayer.set_input(1, inputT1)

network.mark_output(shuffleLayer.get_output(0))
engineString = builder.build_serialized_network(network, config)
engine = trt.Runtime(logger).deserialize_cuda_engine(engineString)
context = engine.create_execution_context()
context.set_shape_input(1, data1)  # 运行时绑定真实形状张量值
_, stream = cudart.cudaStreamCreate()

inputH0 = np.ascontiguousarray(data0.reshape(-1))
inputH1 = np.ascontiguousarray(np.zeros([4], dtype=np.int32).reshape(-1))  # 传形状张量数据可用垃圾值
outputH0 = np.empty(context.get_binding_shape(2), dtype=trt.nptype(engine.get_binding_dtype(2)))
_, inputD0 = cudart.cudaMallocAsync(inputH0.nbytes, stream)
_, inputD1 = cudart.cudaMallocAsync(inputH1.nbytes, stream)
_, outputD0 = cudart.cudaMallocAsync(outputH0.nbytes, stream)

cudart.cudaMemcpyAsync(inputD0, inputH0.ctypes.data, inputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
cudart.cudaMemcpyAsync(inputD1, inputH1.ctypes.data, inputH1.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
context.execute_async_v2([int(inputD0), int(inputD1), int(outputD0)], stream)
cudart.cudaMemcpyAsync(outputH0.ctypes.data, outputD0, outputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
cudart.cudaStreamSynchronize(stream)

print("inputH0 :", data0.shape)
print(data0)
print("inputH1 :", data1.shape)
print(data1)
print("outputH0:", outputH0.shape)
print(outputH0)

cudart.cudaStreamDestroy(stream)
cudart.cudaFree(inputD0)
cudart.cudaFree(inputD1)
cudart.cudaFree(outputD0)
```

- 输出张量形状 (1,4,5,3)，结果与静态 set_input 示例代码相同
- 建立网络时需要 profile，并在运行时绑定真实形状张量的值，否则会有下面几种报错：

```
# 没有用 profile
[TRT] [E] 4: [network.cpp::validate::2919] Error Code 4: Internal Error (Network has dynamic or shape inputs, but no optimization profile has been defined.)
# 没有正确设置 profile
[TRT] [E] 4: [network.cpp::validate::2984] Error Code 4: Internal Error (inputT1: optimization profile is missing values for shape input)
# 没有在运行时绑定形状张量的值
[TRT] [E] 3: [executionContext.cpp::resolveSlots::1481] Error Code 3: API Usage Error (Parameter check failed at: runtime/api/executionContext.cpp::resolveSlots::1481, condition: allInputShapesSpecified(routine)
)
# 绑定的形状张量的值与被 shufle 张量形状不匹配
[TRT] [E] 3: [executionContext.cpp::setInputShapeBinding::1016] Error Code 3: API Usage Error (Parameter check failed at: runtime/api/executionContext.cpp::setInputShapeBinding::1016, condition: data[i] <= profileMaxShape[i]. Supplied binding shapes [2,8,10,6] for bindings[1] exceed min ~ max range at index 1, maximum shape in profile is 5, minimum shape in profile is 1, but supplied shape is 8.

)
```

dynamic shape 模式下的 shuffle + set_input（使用 context.set_binding_shape）

```python
import numpy as np
from cuda import cudart
import tensorrt as trt

nIn, cIn, hIn, wIn = 1, 3, 4, 5
data = np.arange(cIn, dtype=np.float32).reshape(cIn, 1, 1) * 100 + np.arange(hIn).reshape(1, hIn, 1) * 10 + np.arange(wIn).reshape(1, 1, wIn)
data = data.reshape(nIn, cIn, hIn, wIn).astype(np.float32)

np.set_printoptions(precision=8, linewidth=200, suppress=True)
cudart.cudaDeviceSynchronize()

logger = trt.Logger(trt.Logger.ERROR)
builder = trt.Builder(logger)
network = builder.create_network(1 << int(trt.NetworkDefinitionCreationFlag.EXPLICIT_BATCH))
profile = builder.create_optimization_profile()  # 需要使用 profile
config = builder.create_builder_config()
config.max_workspace_size = 1 << 30
inputT0 = network.add_input('inputT0', trt.DataType.FLOAT, (-1, -1, -1, -1))
profile.set_shape(inputT0.name, (1, 1, 1, 1), (nIn, cIn, hIn, wIn), (nIn * 2, cIn * 2, hIn * 2, wIn * 2))
config.add_optimization_profile(profile)

oneLayer = network.add_constant([1], np.array([1], dtype=np.int32))
shape0Layer = network.add_shape(inputT0)
shape1Layer = network.add_concatenation([shape0Layer.get_output(0), oneLayer.get_output(0)])
shape1Layer.axis = 0

shuffleLayer = network.add_shuffle(inputT0)  # 给 inputT0 的末尾加上一维 1
shuffleLayer.set_input(1, shape1Layer.get_output(0))
#shuffleLayer = network.add_shuffle(inputT0)                                                    # 错误的做法，因为 dynamic shape 模式下 inputT0.shape 可能含有多于 1 个 -1，不能作为新形状
#shuffleLayer.reshape_dims = tuple(inputT0.shape) + (1,)

shape2Layer = network.add_shape(shuffleLayer.get_output(0))
shape3Layer = network.add_slice(shape2Layer.get_output(0), [0], [4], [1])
```

```
shuffle2Layer = network.add_shuffle(shuffleLayer.get_output(0))  # 把新加上去的最后一维 1 去掉（set_input 的参数也可直接用 shape0Layer.get_output(0)）
shuffle2Layer.set_input(1, shape3Layer.get_output(0))
#shuffle2Layer = network.add_shuffle(shuffleLayer.get_output(0))                                    # 错误的做法，理由同上
#shuffle2Layer.reshape_dims = tuple(shuffleLayer.get_output(0))[:-1]

network.mark_output(shuffleLayer.get_output(0))
network.mark_output(shuffle2Layer.get_output(0))

engineString = builder.build_serialized_network(network, config)
engine = trt.Runtime(logger).deserialize_cuda_engine(engineString)
context = engine.create_execution_context()
context.set_binding_shape(0, data.shape)
_, stream = cudart.cudaStreamCreate()

inputH0 = np.ascontiguousarray(data.reshape(-1))
outputH0 = np.empty(context.get_binding_shape(1), dtype=trt.nptype(engine.get_binding_dtype(1)))
outputH1 = np.empty(context.get_binding_shape(2), dtype=trt.nptype(engine.get_binding_dtype(2)))
_, inputD0 = cudart.cudaMallocAsync(inputH0.nbytes, stream)
_, outputD0 = cudart.cudaMallocAsync(outputH0.nbytes, stream)
_, outputD1 = cudart.cudaMallocAsync(outputH1.nbytes, stream)

cudart.cudaMemcpyAsync(inputD0, inputH0.ctypes.data, inputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
context.execute_async_v2([int(inputD0), int(outputD0), int(outputD1)], stream)
cudart.cudaMemcpyAsync(outputH0.ctypes.data, outputD0, outputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
cudart.cudaMemcpyAsync(outputH1.ctypes.data, outputD1, outputH1.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
cudart.cudaStreamSynchronize(stream)

print("inputH0 :", data.shape)  # 只打印形状
#print(data)
print("outputH0:", outputH0.shape)
#print(outputH0)
print("outputH1:", outputH1.shape)
#print(outputH1)

cudart.cudaStreamDestroy(stream)
cudart.cudaFree(inputD0)
cudart.cudaFree(outputD0)
cudart.cudaFree(outputD1)
```

- 输出结果

```
inputH0 : (1, 3, 4, 5)
outputH0: (1, 3, 4, 5, 1)
outputH1: (1, 3, 4, 5)
```