TE API 参考

文档版本	01
发布日期	2020-05-09

华为技术有限公司

商标声明

HUAWEI和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

目 录

1 TE 简介

TE（Tensor Engine）是基于TVM（Tensor Virtual Machine）的自定义算子开发框架。TVM是社区的开源项目，旨在将各算子的生成规则进一步抽象，将算子本身分成各个操作原语，在需要的时候加以组合。TVM会根据算子的计算过程的定义，使用Schedule技术和Codegen技术，生成对指定硬件的算子。

由于Schedule是描述在硬件上实现一个算子的计算过程，这需要较强的硬件知识。为了简化用户书写算子的难度，我们在TVM的基础上，简化了书写Schedule的难度，采用“Auto schedule”的概念，提供了一组TensorEngine API，来组合出算子的计算。用户通过使用API进行适当的组合定义一个算子的计算过程，把Schedule交给Auto schedule去完成。本文介绍在TVM基础上定义的微过程API，用户可通过这些API来开发自己的算子。

2 说明

为了改善用户自定义算子的易用性，提高开发效率，我们对部分Vector算子进行了模块化封装。针对Element wise操作的接口，用户需要利用TVM原语来定义自己的输入Tensor（张量），然后调用封装的接口，简要描述下自定义算子的计算过程，然后再调用提供的Auto schedule和Build接口把自定义算子编译生成二进制可执行文件。

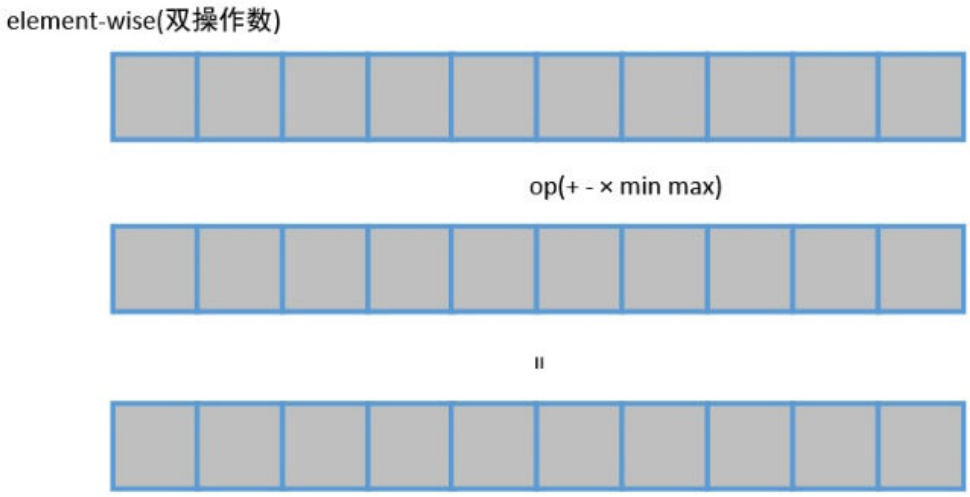
在Tensor Engine中提供了一组封装好的接口，目前主要涵盖向量运算，包括Element-wise类操作接口、Reduction 操作接口、Broadcast操作接口、Index操作接口、Concat操作接口、卷积接口、4D/5D互转接口、矩阵乘接口。

您可以在DDK包的安装目录下的“ddk/site-packages/te-0.4.0.egg/te/lang/cce/te_compute”、“ddk/site-packages/te-0.4.0.egg/te/lang/cce/te_schedule”和“ddk/site-packages/topi-0.4.0.egg/topi/generic”目录下查看接口的定义文件。如果通过引导安装的方式同时安装Mind Studio和DDK，您可以使用Mind Studio安装用户登录Mind Studio服务器，在“~/tools/che/ddk/ddk/site-packages”路径下的对应目录下查看接口定义文件。每个接口对应的定义文件请参见下文中具体的接口描述。关于如何使用这些接口进行自定义算子的开发以及代码的编译运行指导，请参见《TE自定义算子开发指导》。

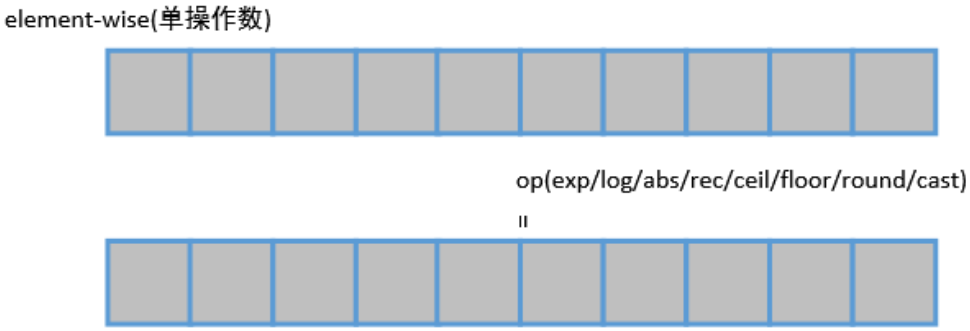
Element-wise 类操作接口

对输入数据进行逐个元素对应运算的操作，这种操作的输出往往具有与输入相同的形状。

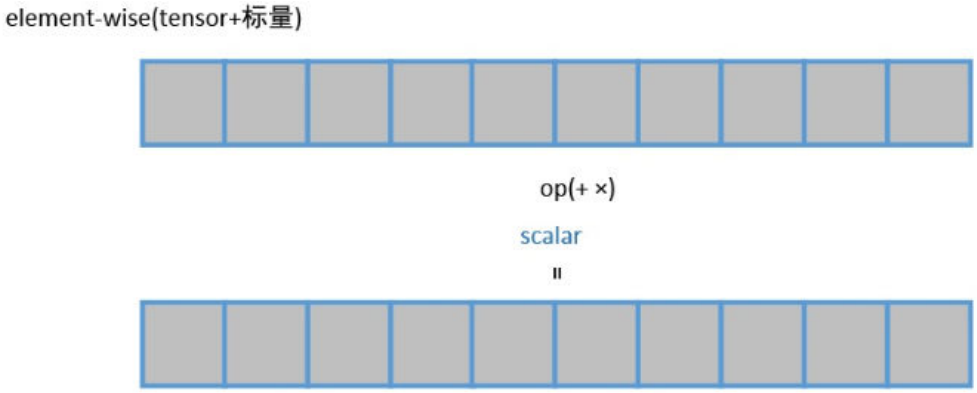
- 双操作数操作。输入两个Tensor（张量）进行2个Tensor之间逐元素运算，得到一个结果的操作。
  - 两个Tensor对应元素相加，详细请参见**3.1 te.lang.cce.vadd(lhs, rhs)**。
  - 两个Tensor对应元素相减，详细请参见**3.2 te.lang.cce.vsub(lhs, rhs)**。
  - 两个Tensor对应元素相乘，详细请参见**3.3 te.lang.cce.vmul(lhs, rhs)**。
  - 两个Tensor对应元素比较取较小值，详细请参见**3.4 te.lang.cce.vmin(lhs, rhs)**。
  - 两个Tensor对应元素比较取较大值，详细请参见**3.5 te.lang.cce.vmax(lhs, rhs)**。
  - 两个Tensor对应元素按位求或，详细请参见**3.6 te.lang.cce.vor(lhs, rhs)**。
  - 两个Tensor对应元素按位求与，详细请参见**3.7 te.lang.cce.vand(lhs, rhs)**。



- 单操作数操作。对输入的一个张量进行逐元素运算。
  - 自然指数，详细请参见**3.11 te.lang.cce.vexp(raw_tensor)**。
  - 对数，详细请参见**3.10 te.lang.cce.vlog(raw_tensor)**。
  - 绝对值，详细请参见**3.12 te.lang.cce.vabs(raw_tensor)**。
  - 倒数，详细请参见**3.13 te.lang.cce.vrec(raw_tensor)**。
  - 向上取整，详细请参见**3.21 te.lang.cce.ceil(raw_tensor)**。
  - 向下取整，详细请参见**3.22 te.lang.cce.floor(raw_tensor)**。
  - 四舍六入，详细请参见**3.23 te.lang.cce.round(raw_tensor)**。
  - 转换数据类型，详细请参见**3.14 te.lang.cce.cast_to(data, dtype, f1628IntegerFlag=False)**。
  - 实现relu计算，详细请参见**3.15 te.lang.cce.vrelu(raw_tensor)**。
  - 按位取反，详细请参见**3.16 te.lang.cce.vnot(raw_tensor)**。



- Tensor与标量数值的运算操作。输入Tensor的每个元素与同一个数值进行运算。
  - Tensor加上标量，详细请参见**3.8 te.lang.cce.vadds(raw_tensor, scalar)**。
  - Tensor乘上标量，详细请参见**3.9 te.lang.cce.vmuls(raw_tensor, scalar)**。

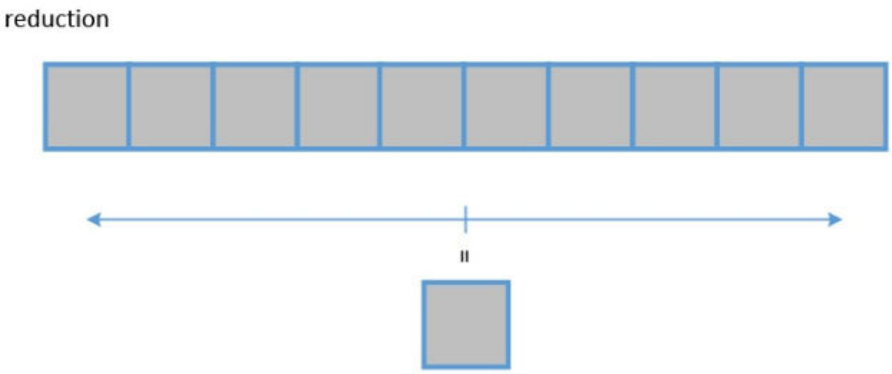


- 三操作数。输入为3个。tensor与标量按照上边的规则混合运算得到一个结果的操作。
  - 对一个Tensor进行缩放后和第二个Tensor相加，详细请参见**3.17 te.lang.cce.vaxpy(lhs, rhs, scalar)**。
  - 输入为三个Tensor，计算x * y + z，详细请参见**3.18 te.lang.cce.vmla(x, y, z)**。
  - 输入为三个Tensor，计算x * z + y，详细请参见**3.19 te.lang.cce.vmadd(x, y, z)**。
  - 输入为三个Tensor，计算relu (x * z + y)，详细请参见**3.20 te.lang.cce.vmaddrelu(x, y, z)**。

Reduction 操作接口

压缩某一维的数据，沿着指定方向将数据进行累加或累乘等操作，该操作输出比输入数据维度少一维的结果。

- 沿某个轴进行累加，详细请参见**3.24 te.lang.cce.sum(raw_tensor, axis, keepdims=False)**。
- 沿某个轴求最小值，详细请参见**3.25 te.lang.cce.reduce_min(raw_tensor, axis, keepdims=False)**。
- 沿某个轴求最大值，详细请参见**3.26 te.lang.cce.reduce_max(raw_tensor, axis, keepdims=False)**。



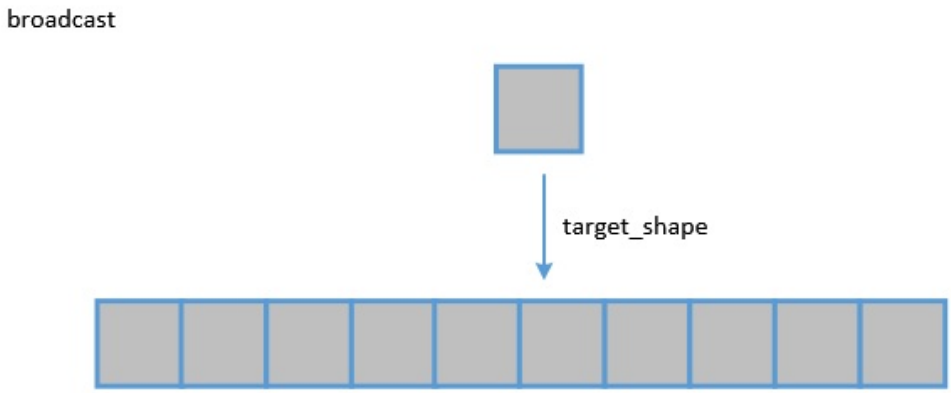
须知

使用限制：由于CCE计算平台的数据排布限制，reduction操作后的数据需要进行一次重排才能进行后续的操作。故当前在使用不同类型的接口时，暂时不能在reduction操作后进行任何向量运算操作。

Broadcast 操作接口

Broadcast操作主要是用于处理两个不同形状的tensor进行计算，将低维度的一个操作数按照高维度操作数的维度进行广播，使得两个操作数的维度相同后，再进行element-wise计算。

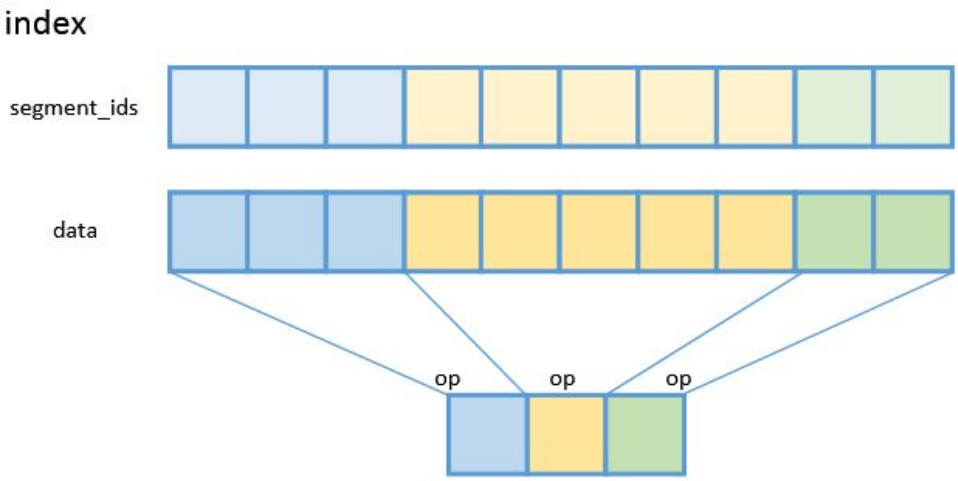
把较小的tensor广播为较大的tensor，详细请参见**3.28 te.lang.cce.broadcast(var, shape, output_dtype=None)**。



Index 操作接口

Index操作，对Tensor进行分段计算，比如求和、均值、内积、最大值、最小值等。

- 对Tensor进行分段求和，详细请参见**3.29 te.lang.cce.unsorted_segment_sum(tensor, segment_ids, num_segments, init_value=0)**。
- 对Tensor进行分段求均值，详细请参见**3.30 te.lang.cce.unsorted_segment_mean(tensor, segment_ids, num_segments, init_value=0)**。
- 对Tensor进行分段求内积，详细请参见**3.31 te.lang.cce.unsorted_segment_prod(tensor, segment_ids, num_segments, init_value=0)**。
- 对Tensor进行分段求最小值，详细请参见**3.32 te.lang.cce.unsorted_segment_min(tensor, segment_ids, num_segments, init_value=0)**。
- 对Tensor进行分段求最大值，详细请参见**3.33 te.lang.cce.unsorted_segment_max(tensor, segment_ids, num_segments, init_value=0)**。



Concat 操作接口

Concat操作是沿着某个轴对多个输入的Tensor进行连接。

沿一个轴连接Tensor，详细请参见**3.34 te.lang.cce.concat(raw_tensors, axis)**。

卷积接口

卷积接口是用来实现卷积算子的接口。

实现卷积，详细请参见**3.35 te.lang.cce.conv(*args)**。

4D/5D 互转接口

4D维度NCHW与5D维度NC1HWC0互转接口。

- 4D转5D，详细请参见**3.36 te.lang.cce.compute_four2five(input, raw_shape_4D)**。
- 5D转4D，详细请参见**3.37 te.lang.cce.compute_five2four(input, raw_shape_4D)**。

矩阵乘接口

矩阵乘接口是用来实现矩阵相乘的接口。

详细请参见**3.38 te.lang.cce.matmul(tensor_a, tensor_b, trans_a=False, trans_b=False, alpha_num=1.0, beta_num=0.0, tensor_c=None)**。

融合算子接口

TE算子融合功能需要对单算子的融合能力和入参等信息做预处理，因此在生成单算子时需要调用指定API生成单算子的*.so文件和*.json文件。

详细请参见**5.1 bool BuildTeCustomOp(std::string ddkVer, std::string opName, std::string opPath, std::string opFuncName, const char *format, ...)**。

3 compute 接口

3.1 te.lang.cce.vadd(lhs, rhs)

两个tensor按元素相加，元素的数据类型需要一致，支持的类型：float16，float32，int32。int8，uint8会被转换为float16。

该接口在elewise_compute.py中定义。

参数说明

- lhs：左tensor，tvm.tensor类型。
- rhs：右tensor，tvm.tensor类型。

返回值

res_tensor：表示lhs + rhs，tvm.tensor类型

调用示例

```
shape = (1024,1024)
input_dtype = "float16"
data1 = tvm.placeholder(shape, name="data1", dtype=input_dtype)
data2 = tvm.placeholder(shape, name="data2", dtype=input_dtype)
res = te.lang.cce.vadd(data1, data2)
```

3.2 te.lang.cce.vsub(lhs, rhs)

两个tensor按元素相减，元素的数据类型需要一致，支持的类型：float16，float32，int32。int8，uint8会被转换为float16。

该接口在elewise_compute.py中定义。

参数说明

- lhs：左tensor，tvm.tensor类型。
- rhs：右tensor，tvm.tensor类型。

返回值

res_tensor：表示 lhs - rhs，tvm.tensor类型

调用示例

```
shape = (1024,1024)
input_dtype = "float16"
data1 = tvm.placeholder(shape, name="data1", dtype=input_dtype)
data2 = tvm.placeholder(shape, name="data2", dtype=input_dtype)
res = te.lang.cce.vsub(data1, data2)
```

3.3 te.lang.cce.vmul(lhs, rhs)

两个tensor按元素相乘，元素的数据类型需要一致，支持的类型：float16，float32，int32。int8，uint8会被转换为float16。

该接口在elewise_compute.py中定义。

参数说明

- lhs：左tensor，tvm.tensor类型。
- rhs：右tensor，tvm.tensor类型。

返回值

res_tensor：表示lhs * rhs，tvm.tensor类型

调用示例

```
shape = (1024,1024)
input_dtype = "float16"
data1 = tvm.placeholder(shape, name="data1", dtype=input_dtype)
data2 = tvm.placeholder(shape, name="data2", dtype=input_dtype)
res = te.lang.cce.vmul(data1, data2)
```

3.4 te.lang.cce.vmin(lhs, rhs)

两个tensor按元素比较并取较小值，元素的数据类型需要一致，支持的类型：float16，float32，int32。int8，uint8会被转换为float16。

该接口在elewise_compute.py中定义。

参数说明

- lhs：左tensor，tvm.tensor类型。
- rhs：右tensor，tvm.tensor类型。

返回值

res_tensor：结果tensor，tvm.tensor类型

调用示例

```
shape = (1024,1024)
input_dtype = "float16"
data1 = tvm.placeholder(shape, name="data1", dtype=input_dtype)
data2 = tvm.placeholder(shape, name="data2", dtype=input_dtype)
res = te.lang.cce.vmin(data1, data2)
```

3.5 te.lang.cce.vmax(lhs, rhs)

两个tensor按元素比较并取较大值，元素的数据类型需要一致，支持的类型：float16，float32，int32。int8，uint8会被转换为float16。

该接口在elewise_compute.py中定义。

参数说明

- lhs：左tensor，tvm.tensor类型。
- rhs：右tensor，tvm.tensor类型。

返回值

res_tensor：结果tensor，tvm.tensor类型

调用示例

```
shape = (1024,1024)
input_dtype = "float16"
data1 = tvm.placeholder(shape, name="data1", dtype=input_dtype)
data2 = tvm.placeholder(shape, name="data2", dtype=input_dtype)
res = te.lang.cce.vmax(data1, data2)
```

3.6 te.lang.cce.vor(lhs, rhs)

两个tensor元素按位取或，元素的数据类型需要一致，支持的类型：int16，uint16。

该接口在elewise_compute.py中定义。

参数说明

- lhs：左tensor，tvm.tensor类型。
- rhs：右tensor，tvm.tensor类型。

返回值

res_tensor：表示lhs按位或rhs，tvm.tensor类型

调用示例

```
shape = (1024,1024)
input_dtype = "int16"
data1 = tvm.placeholder(shape, name="data1", dtype=input_dtype)
data2 = tvm.placeholder(shape, name="data2", dtype=input_dtype)
res = te.lang.cce.vor(data1, data2)
```

3.7 te.lang.cce.vand(lhs, rhs)

两个tensor元素按位取与，元素的数据类型需要一致，支持的类型：int16，uint16。

该接口在elewise_compute.py中定义。

参数说明

- lhs：左tensor，tvm.tensor类型。
- rhs：右tensor，tvm.tensor类型。

返回值

res_tensor：表示lhs按位与rhs，tvm.tensor类型

调用示例

```
shape = (1024,1024)
input_dtype = "int16"
data1 = tvm.placeholder(shape, name="data1", dtype=input_dtype)
data2 = tvm.placeholder(shape, name="data2", dtype=input_dtype)
res = te.lang.cce.vand(data1, data2)
```

3.8 te.lang.cce.vadds(raw_tensor, scalar)

将raw_tensor中每个元素加上标量scalar，支持的类型：float16，float32。int8，uint8，int32会被转换为float16。如果scalar数据类型与raw_tensor数据类型不一致，计算中会转换为对应数据类型。

该接口在elewise_compute.py中定义。

参数说明

- raw_tensor：输入tensor，tvm.tensor类型。
- scalar：raw_tensor中元素要加的系数，标量类型。

返回值

res_tensor：表示raw_tensor + scalar，tvm.tensor类型

调用示例

```
shape = (1024,1024)
input_dtype = "float16"
```

```
data = tvm.placeholder(shape, name="data", dtype=input_dtype)
scalar = tvm.const(2, dtype =input_dtype)
res = te.lang.cce.vadds(data, scalar)
```

3.9 te.lang.cce.vmuls(raw_tensor, scalar)

将raw_tensor中每个元素乘上标量scalar，支持的类型：float16，float32。int8，uint8，int32会被转换为float16。如果scalar数据类型与raw_tensor数据类型不一致，计算中会转换为对应数据类型。

该接口在elewise_compute.py中定义。

参数说明

- raw_tensor：输入tensor，tvm.tensor类型。
- scalar：raw_tensor中元素要乘的系数，标量类型。

返回值

res_tensor：表示raw_tensor * scalar，tvm.tensor类型

调用示例

```
shape = (1024,1024)
input_dtype = "float16"
data = tvm.placeholder(shape, name="data", dtype=input_dtype)
scalar = tvm.const(2, dtype =input_dtype)
res = te.lang.cce.vmuls(data, scalar)
```

3.10 te.lang.cce.vlog(raw_tensor)

对raw_tensor中的每个元素做对数ln(x)运算，支持的类型：float16。int8，uint8，int32，float32类型会被转换为float16。

该接口在elewise_compute.py中定义。

参数说明

raw_tensor：输入tensor，tvm.tensor类型。

返回值

res_tensor：表示ln(raw_tensor)，tvm.tensor类型。

调用示例

```
shape = (1024,1024)
input_dtype = "float16"
data = tvm.placeholder(shape, name="data", dtype=input_dtype)
res = te.lang.cce.vlog(data)
```

3.11 te.lang.cce.vexp(raw_tensor)

对tensor中的每个元素做自然指数运算e^x，支持的类型：float16。int8，uint8，int32，float32类型会被转换为float16。

该接口在elewise_compute.py中定义。

参数说明

raw_tensor：输入tensor，tvm.tensor类型。

返回值

res_tensor：表示e^(raw_tensor)，tvm.tensor类型。

调用示例

```
shape = (1024,1024)
input_dtype = "float16"
data = tvm.placeholder(shape, name="data", dtype=input_dtype)
res = te.lang.cce.vexp(data)
```

3.12 te.lang.cce.vabs(raw_tensor)

对tensor中的每个元素做绝对值运算|x|，支持的类型：float16。int8，uint8，int32，float32类型会被转换为float16。

该接口在elewise_compute.py中定义。

参数说明

raw_tensor：输入tensor，tvm.tensor类型。

返回值

res_tensor：表示|raw_tensor|，tvm.tensor类型。

调用示例

```
shape = (1024,1024)
input_dtype = "float16"
data = tvm.placeholder(shape, name="data", dtype=input_dtype)
res = te.lang.cce.vabs(data)
```

3.13 te.lang.cce.vrec(raw_tensor)

对tensor中的每个元素做倒数运算 1 / x，支持的类型：float16，float32。int8，uint8，int32类型会被转换为float16。

该接口在elewise_compute.py中定义。

参数说明

raw_tensor：输入tensor，tvm.tensor类型。

返回值

res_tensor：表示1 / raw_tensor，tvm.tensor类型。

调用示例

```
shape = (1024,1024)
input_dtype = "float16"
data = tvm.placeholder(shape, name="data", dtype=input_dtype)
res = te.lang.cce.vrec(data)
```

3.14 te.lang.cce.cast_to(data, dtype, f1628IntegerFlag=False)

数据类型转换，把data中的数据转换为dtype类型。

该接口在common.py中定义。

支持如下的类型转换。

表 3-1 支持的类型转换

源数据类型	目的数据类型
float32	float16
float32	int8
float32	uint8
float16	float32
float16	int8
float16	uint8
float16	int32
int8	float16
int8	uint8
int32	float16
int32	int8
int32	uint8

参数说明

- data：输入tensor，tvm.tensor类型。
- dtype：目的数据类型，字符串类型。
- f1628IntegerFlag：默认值为False。如果转换前数据的小数部分为0，参数f1628IntegerFlag需要设为True；如果转换前数据的小数部分不为0，参数f1628IntegerFlag需要设为False。

返回值

res_tensor：转换后的数据，tvm.tensor类型。

调用示例

```
shape = (1024,1024)
input_dtype = "float16"
data = tvm.placeholder(shape, name="data", dtype=input_dtype)
res = te.lang.cce.cast_to(data,"float32")
```

3.15 te.lang.cce.vrelu(raw_tensor)

对tensor中的每个元素做线性整流运算relu，支持的类型：float16。int8，uint8，int32，float32类型会被转换为float16。

该接口在elewise_compute.py中定义。

参数说明

raw_tensor：输入tensor，tvm.tensor类型。

返回值

res_tensor：表示relu(raw_tensor)，tvm.tensor类型。

调用示例

```
shape = (1024,1024)
input_dtype = "float16"
data = tvm.placeholder(shape, name="data", dtype=input_dtype)
res = te.lang.cce.vrelu(data)
```

3.16 te.lang.cce.vnot(raw_tensor)

对tensor中的每个元素按位取反，支持的类型：int16，uint16。

该接口在elewise_compute.py中定义。

参数说明

raw_tensor：输入tensor，tvm.tensor类型。

返回值

res_tensor：表示raw_tensor按位取反，tvm.tensor类型。

调用示例

```
shape = (1024,1024)
input_dtype = "int16"
data = tvm.placeholder(shape, name="data", dtype=input_dtype)
res = te.lang.cce.vnot(data)
```

3.17 te.lang.cce.vaxpy(lhs, rhs, scalar)

将lhs中每个元素乘上标量scalar，再加上rhs中的对应元素，lhs和rhs tensor的数据类型要求一致，支持的类型：float16、float32。int8、uint8、int32类型会被转换为float16。

scalar的数据类型如果跟tensor不一致，会被转换成tensor的数据类型。

该接口在elewise_compute.py中定义。

参数说明

- lhs：左tensor，tvm.tensor类型。
- rhs：右tensor，tvm.tensor类型。
- scalar：lhs中元素要乘的系数，标量类型。

返回值

res_tensor：表示lhs * scalar + rhs，tvm.tensor类型。

调用示例

```
shape = (1024,1024)
input_dtype = "float16"
data1 = tvm.placeholder(shape, name="data1", dtype=input_dtype)
data2 = tvm.placeholder(shape, name="data2", dtype=input_dtype)
scalar = tvm.const(2, dtype =input_dtype)
res = te.lang.cce.vaxpy(data1, data2, scalar)
```

3.18 te.lang.cce.vmla(x, y, z)

将x中每个元素乘上y中的对应元素，再加上z中的对应元素，三个tensor的数据类型要求一致，支持的类型：float16、float32。int8、uint8、int32类型会被转换为float16。

该接口在elewise_compute.py中定义。

参数说明

- x：tensor，tvm.tensor类型
- y：tensor，tvm.tensor类型
- z：tensor，tvm.tensor类型

返回值

res_tensor：表示x * y + z，tvm.tensor类型

调用示例

```
shape = (1024,1024)
input_dtype = "float16"
data1 = tvm.placeholder(shape, name="data1", dtype=input_dtype)
data2 = tvm.placeholder(shape, name="data2", dtype=input_dtype)
data3 = tvm.placeholder(shape, name="data3", dtype=input_dtype)
res = te.lang.cce.vmla(data1, data2, data3)
```

3.19 te.lang.cce.vmadd(x, y, z)

将x中每个元素乘上z中的对应元素，再加上y中的对应元素，三个tensor的数据类型要求一致，支持的类型：float16、float32。int8、uint8、int32类型会被转换为float16。

该接口在elewise_compute.py中定义。

参数说明

- x：tensor，tvm.tensor类型。
- y：tensor，tvm.tensor类型。
- z：tensor，tvm.tensor类型。

返回值

res_tensor：表示x * z + y，tvm.tensor类型。

调用示例

```
shape = (1024,1024)
input_dtype = "float16"
data1 = tvm.placeholder(shape, name="data1", dtype=input_dtype)
data2 = tvm.placeholder(shape, name="data2", dtype=input_dtype)
data3 = tvm.placeholder(shape, name="data3", dtype=input_dtype)
res = te.lang.cce.vmadd(data1, data2, data3)
```

3.20 te.lang.cce.vmaddrelu(x, y, z)

将x中每个元素乘上z中的对应元素，再加上y中的对应元素，然后做线性整流，三个tensor的数据类型要求一致，支持的类型：float16、float32。int8、uint8、int32类型会被转换为float16。

该接口在elewise_compute.py中定义。

参数说明

- x：tensor，tvm.tensor类型。
- y：tensor，tvm.tensor类型。
- z：tensor，tvm.tensor类型。

返回值

res_tensor：表示relu(x * z + y)，tvm.tensor类型。

调用示例

```
shape = (1024,1024)
input_dtype = "float16"
data1 = tvm.placeholder(shape, name="data1", dtype=input_dtype)
data2 = tvm.placeholder(shape, name="data2", dtype=input_dtype)
data3 = tvm.placeholder(shape, name="data3", dtype=input_dtype)
res = te.lang.cce.vmaddrelu(data1, data2, data3)
```

3.21 te.lang.cce.ceil(raw_tensor)

对raw_tensor中的每个元素向上取整，支持的类型：float16。float32类型会被转换为float16。结果为int32。

该接口在cast_compute.py中定义。

参数说明

raw_tensor：输入tensor，tvm.tensor类型。

返回值

res_tensor：表示ceil(raw_tensor)，tvm.tensor类型。

调用示例

```
shape = (1024,1024)
input_dtype = "float16"
data = tvm.placeholder(shape, name="data", dtype=input_dtype)
res = te.lang.cce.ceil(data)
```

3.22 te.lang.cce.floor(raw_tensor)

对raw_tensor中的每个元素向下取整，支持的类型：float16。float32类型会被转换为float16。结果为int32。

该接口在cast_compute.py中定义。

参数说明

raw_tensor：输入tensor，tvm.tensor类型

返回值

res_tensor：表示floor(raw_tensor)，tvm.tensor类型

调用示例

```
shape = (1024,1024)
input_dtype = "float16"
data = tvm.placeholder(shape, name="data", dtype=input_dtype)
res = te.lang.cce.floor(data)
```

3.23 te.lang.cce.round(raw_tensor)

对raw_tensor中的每个元素四舍六入，0.5取偶数，例如`1.5->2.0，2.5->2.0`，支持的类型：float16。float32类型会被转换为float16。结果为int32。

该接口在cast_compute.py中定义。

参数说明

raw_tensor：输入tensor，tvm.tensor类型

返回值

res_tensor：表示round(raw_tensor)，tvm.tensor类型

调用示例

```
shape = (1024,1024)
input_dtype = "float16"
```

```
data = tvm.placeholder(shape, name="data", dtype=input_dtype)
res = te.lang.cce.round(data)
```

3.24 te.lang.cce.sum(raw_tensor, axis, keepdims=False)

按某个轴求和，进行降维，支持的类型：float16、float32。int8、uint8、int32类型会被转换为float16。

该接口在reduction_compute.py中定义。

参数说明

- raw_tensor：输入tensor，tvm.tensor类型
- axis：做reduce操作的轴，取值范围：[-d,d-1]，其中d是raw_tensor的维数，int或list类型
- keepdims：默认值是False，表示做reduce操作后，操作的轴长度为0，例如，原shape是(10,10,10)，keepdims=False时，reduce后shape是(10,10)。若将该参数值设置为True，表示做reduce操作后，操作的轴的长度设置为1，例如，原shape是(10,10,10)，keepdims=True时，reduce后shape是(10,10,1)。

返回值

res_tensor：求和后的tensor，tvm.tensor类型

调用示例

```
shape = (1024,1024)
input_dtype = "float16"
data = tvm.placeholder(shape, name="data", dtype=input_dtype)
res = te.lang.cce.sum(data, axis=1)
```

3.25 te.lang.cce.reduce_min(raw_tensor, axis, keepdims=False)

按某个轴求最小值，进行降维，支持的类型：float16。int8、uint8、int32、float32类型会被转换为float16。

该接口在reduction_compute.py中定义。

参数说明

- raw_tensor：输入tensor，tvm.tensor类型。
- axis：做reduce操作的轴，取值范围：[-d,d-1]，其中d是raw_tensor的维数，int或list类型。
- keepdims：默认值是False，表示做reduce操作后，操作的轴长度为0，例如，原shape是(10,10,10)，keepdims=False时，reduce后shape是(10,10)。若将该参数值设置为True，表示做reduce操作后，操作的轴的长度设置为1，例如，原shape是(10,10,10)，keepdims=True时，reduce后shape是(10,10,1)。

返回值

res_tensor：取最小值后的tensor，tvm.tensor类型。

调用示例

```
shape = (1024,1024)
input_dtype = "float16"
data = tvm.placeholder(shape, name="data", dtype=input_dtype)
res = te.lang.cce.reduce_min(data, axis=1)
```

3.26 te.lang.cce.reduce_max(raw_tensor, axis, keepdims=False)

按某个轴求最大值，进行降维，支持的类型：float16。int8、uint8、int32、float32类型会被转换为float16。

该接口在reduction_compute.py中定义。

参数说明

- raw_tensor：输入tensor，tvm.tensor类型。
- axis：做reduce操作的轴，取值范围：[-d,d-1]，其中d是raw_tensor的维数，int或list类型。
- keepdims：默认值是False，表示做reduce操作后，操作的轴长度为0，例如，原shape是(10,10,10)，keepdims=False时，reduce后shape是(10,10)。若将该参数值设置为True，表示做reduce操作后，操作的轴的长度设置为1，例如，原shape是(10,10,10)，keepdims=True时，reduce后shape是(10,10,1)。

返回值

res_tensor：取最大值后的tensor，tvm.tensor类型。

调用示例

```
shape = (1024,1024)
input_dtype = "float16"
data = tvm.placeholder(shape, name="data", dtype=input_dtype)
res = te.lang.cce.reduce_max(data, axis=1)
```

3.27 te.lang.cce.reduce_prod(raw_tensor, axis, keepdims=False)

按某个轴求乘积，进行降维，支持的类型：float16。int8、uint8、int32、float32类型会被转换为float16。

该接口在reduction_compute.py中定义。

参数说明

- raw_tensor：输入tensor，tvm.tensor类型。
- axis：做reduce操作的轴，取值范围：[-d,d-1]，其中d是raw_tensor的维数，int类型。
- keepdims：默认值是False，表示做reduce操作后，操作的轴长度为0，例如，原shape是(10,10,10)，keepdims=False时，reduce后shape是(10,10)。若将该参数值设置为True，表示做reduce操作后，操作的轴的长度设置为1，例如，原shape是(10,10,10)，keepdims=True时，reduce后shape是(10,10,1)。

返回值

res_tensor：按某个轴求乘积后的tensor，tvm.tensor类型。

调用示例

```
shape = (1024,1024)
input_dtype = "float16"
data = tvm.placeholder(shape, name="data", dtype=input_dtype)
res = te.lang.cce.reduce_prod(data, axis=1)
```

3.28 te.lang.cce.broadcast(var, shape, output_dtype=None)

把var broadcast为大小为shape的tensor，结果的数据类型由output_dtype指定，var可以是标量，或者是一个tensor，要求var的shape与第二个参数shape的长度一致，每个维度的大小要么与shape相等，要么为1，为1的维度会被broadcast到与shape一致。例如var的维度为（2，1，64），shape为（2，128，64），运算结果var的维度变为（2，128，64）。支持的类型：float16、float32、int32。

该接口在broadcast_compute.py中定义。

参数说明

- var：需要broadcast的数据，标量或者tensor类型。
- shape：目标shape，进行broadcast操作的目标shape。
- output_dtype：输出数据类型，默认值var.dtype。

返回值

res_tensor：由var扩展后得到的tensor，shape为参数指定的shape，数据类型为output_dtype。

调用示例

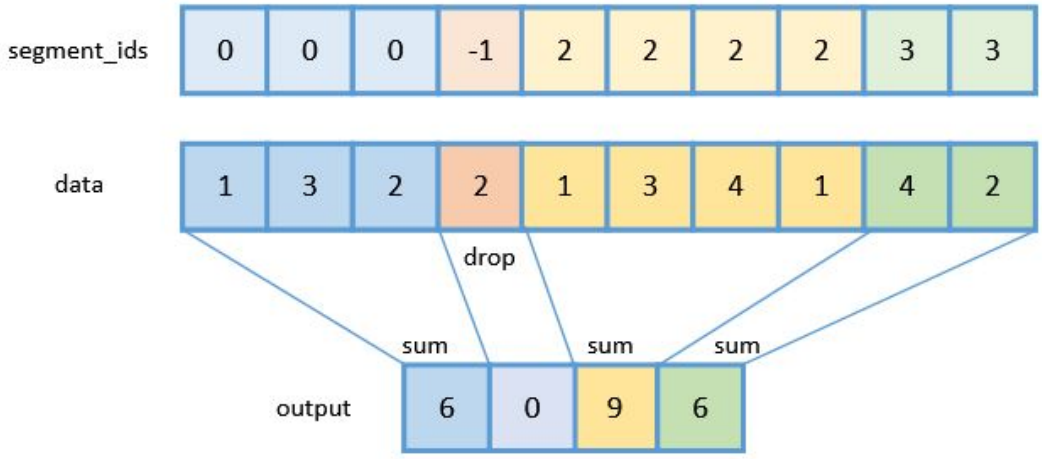
```
outshape = (1024,1024)
shape = (1024,1)
input_dtype = "float16"
data = tvm.placeholder(shape, name="data", dtype=input_dtype)
res = te.lang.cce.broadcast(data, outshape)
```

3.29 te.lang.cce.unsorted_segment_sum(tensor, segment_ids, num_segments, init_value=0)

使用数组segment_ids对tensor进行分段求和。假设输入为data，输出为output，则output[i] = sum(data[j...])，其中“j...”是一个数组，“j...”中的元素j满足：segment_ids[j] == i。

如果某个下标i在segment_ids中没有出现，则output[i] = init_value。比如下图中，1在segment_ids中没有出现，则output[1] = 0。

如果segment_ids中某个值为负数，则对应位置的data中的值则丢弃。比如下图中，segment_ids[3] = -1，则data[3]的值被丢弃，不参与计算。



segment_ids其长度必须和data的第一维的长度相同。num_segments必须大于等于segment_ids的最大值加1。

支持的数据类型：float16、float32、int32。

该接口在segment_compute.py中定义。

参数说明

- tensor：输入tensor，必须是如下数据类型之一：float16、float32、int32。
- segment_ids：一维数组，对输入tensor进行分段的数组，其长度必须和输入tensor的第一维的长度相同。同时支持有序和无序。
- num_segments：输出tensor的第一维的长度。其值必须大于等于segment_ids的最大值加1。
- init_value：当segment_ids中某个下标不存在时，其输出的默认值。根据算子的实现来确定。默认值为0。

返回值

res_tensor：表示计算后的tensor。

调用示例

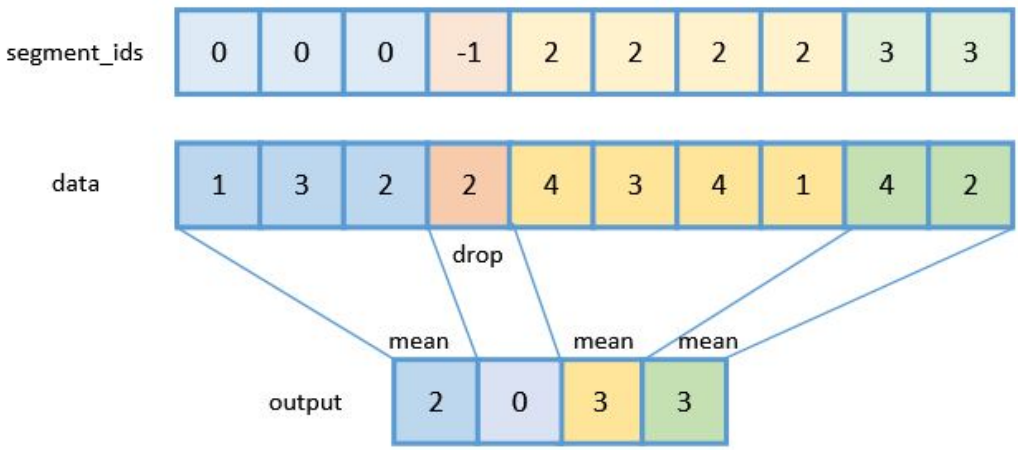
```
import tvm
import te.lang.cce
shape = (5,1024)
input_dtype = "float16"
data = tvm.placeholder(shape, name="data1", dtype=input_dtype)
segment_ids = [1,1,4,5,5]
num_segments = 6
res = te.lang.cce.unsorted_segment_sum(data, segment_ids, num_segments)
res.shape = (6,1024)
# res[0] = 0
# res[1] = data[0] + data[1]
# res[2] = 0
# res[3] = 0
# res[4] = data[2]
# res[5] = data[3] + data[4]
```

3.30 te.lang.cce.unsorted_segment_mean(tensor, segment_ids, num_segments, init_value=0)

使用数组segment_ids对tensor进行分段求均值。假设输入为data,输出为output，则output[i] = (1 / len(j...)) sum(data[j...])，其中“j...”是一个数组，“j...”中的元素j满足：segment_ids[j] == i。

如果某个下标i在segment_ids中没有出现,则output[i] = init_value。比如下图中,1在segment_ids中没有出现，则output[1] = 0。

如果segment_ids中某个值为负数，则对应位置的data中的值则丢弃。比如下图中，segment_ids[3] = -1，则data[3]的值被丢弃，不参与计算。



segment_ids其长度必须和data的第一维的长度相同。num_segments必须大于等于segment_ids的最大值加1。

支持的数据类型：float16、float32、int32。

该接口在segment_compute.py中定义。

参数说明

- tensor：输入tensor，必须是如下数据类型之一：float16、float32、int32。
- segment_ids：一维数组，对输入tensor进行分段的数组，其长度必须和输入tensor的第一维的长度相同。同时支持有序和无序。
- num_segments：输出tensor的第一维的长度。其值必须大于等于segment_ids的最大值加1。
- init_value：当segment_ids中某个下标不存在时，其输出的默认值。根据算子的实现来确定。默认值为0。

返回值:

res_tensor：表示计算后的tensor。

调用示例

```
import tvm
import te.lang.cce
```

```
shape = (5,1024)
input_dtype = "float16"
data = tvm.placeholder(shape, name="data1", dtype=input_dtype)
segment_ids = [1,1,5,5,5]
num_segments = 6
res = te.lang.cce.unsorted_segment_mean(data, segment_ids, num_segments)
# res.shape = (6,1024)
# res[0] = 0
# res[1] = (data[0] + data[1]) / 2
# res[2] = 0
# res[3] = 0
# res[4] = 0
# res[5] = (data[2] + data[3] + data[4]) / 3
```

3.31 te.lang.cce.unsorted_segment_prod(tensor, segment_ids, num_segments, init_value=0)

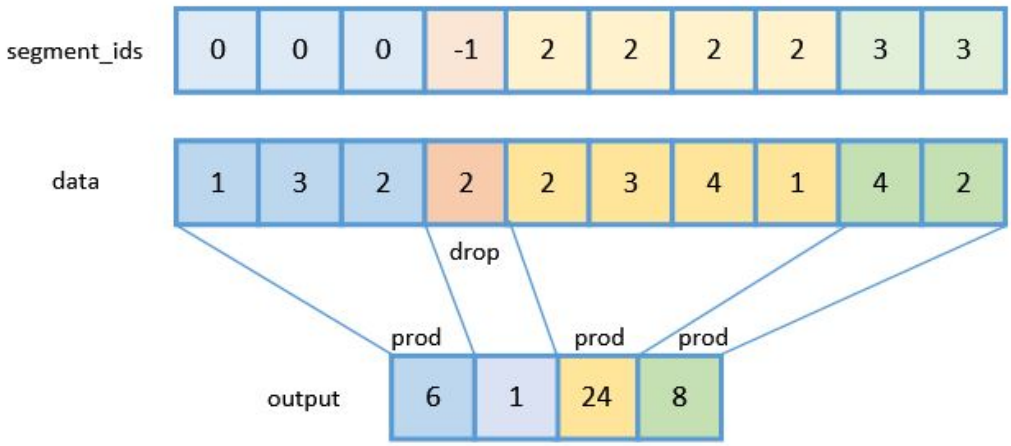
使用数组segment_ids对tensor进行分段求内积。假设输入为data，输出为output，则output[i] = product(data[j...])，其中“j...”是一个数组，“j...”中的元素j满足：segment_ids[j] == i。

说明

product表示求内积，即data[j...]中所有元素相乘。

如果某个下标i在segment_ids中没有出现，则output[i] = init_value。比如下图中，1在segment_ids中没有出现，则output[1] = 0。

如果segment_ids中某个值为负数，则对应位置的data中的值则丢弃。比如下图中，segment_ids[3] = -1，则data[3]的值被丢弃，不参与计算。



segment_ids其长度必须和data的第一维的长度相同。num_segments必须大于等于segment_ids的最大值加1。

支持的数据类型：float16、float32、int32。

该接口在segment_compute.py中定义。

参数说明

- tensor：输入tensor，必须是如下数据类型之一：float16、float32、int32。
- segment_ids：一维数组，对输入tensor进行分段的数组，其长度必须和输入tensor的第一维的长度相同。同时支持有序和无序。

- num_segments：输出tensor的第一维的长度。其值必须大于等于segment_ids的最大值加1。
- init_value：当segment_ids中某个下标不存在时，其输出的默认值。根据算子的实现来确定。默认值为0。

返回值

res_tensor：表示计算后的tensor。

调用示例

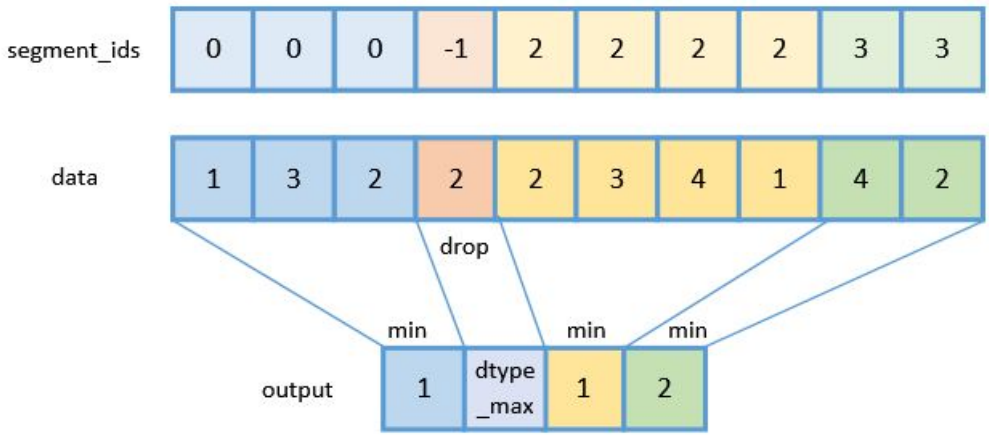
```
import tvm
import te.lang.cce
shape = (5,1024)
input_dtype = "float16"
data = tvm.placeholder(shape, name="data1", dtype=input_dtype)
segment_ids = [1,1,4,5,5]
num_segments = 6
res = te.lang.cce.unsorted_segment_prod(data, segment_ids, num_segments)
# res.shape = (6,1024)
# res[0] = 1
# res[1] = (data[0] * data[1])
# res[2] = 1
# res[3] = 1
# res[4] = data[2]
# res[5] = (data[3] * data[4])
```

3.32 te.lang.cce.unsorted_segment_min(tensor, segment_ids, num_segments, init_value=0)

使用数组segment_ids对tensor进行分段求最小值。假设输入为data，输出为output，则output[i] = min(data[j...])，其中“j...”是一个数组，“j...”中的元素j满足：segment_ids[j] == i。

如果某个下标i在segment_ids中没有出现，则output[i] = init_value。比如下图中，1在segment_ids中没有出现，则output[1] = 0。

如果segment_ids中某个值为负数,则对应位置的data中的值则丢弃。比如下图中，segment_ids[3] = -1，则data[3]的值被丢弃，不参与计算。



segment_ids其长度必须和data的第一维的长度相同。num_segments必须大于等于segment_ids的最大值加1。

支持的数据类型：float16、float32。

该接口在segment_compute.py中定义。

参数说明

- tensor：输入tensor，必须是如下数据类型之一：float16、float32、int32。
- segment_ids：一维数组，对输入tensor进行分段的数组，其长度必须和输入tensor的第一维的长度相同。同时支持有序和无序。
- num_segments：输出tensor的第一维的长度。其值必须大于等于segment_ids的最大值加1。
- init_value：当segment_ids中某个下标不存在时，其输出的默认值。根据算子的实现来确定。默认值为0。

返回值

res_tensor：表示计算后的tensor。

调用示例

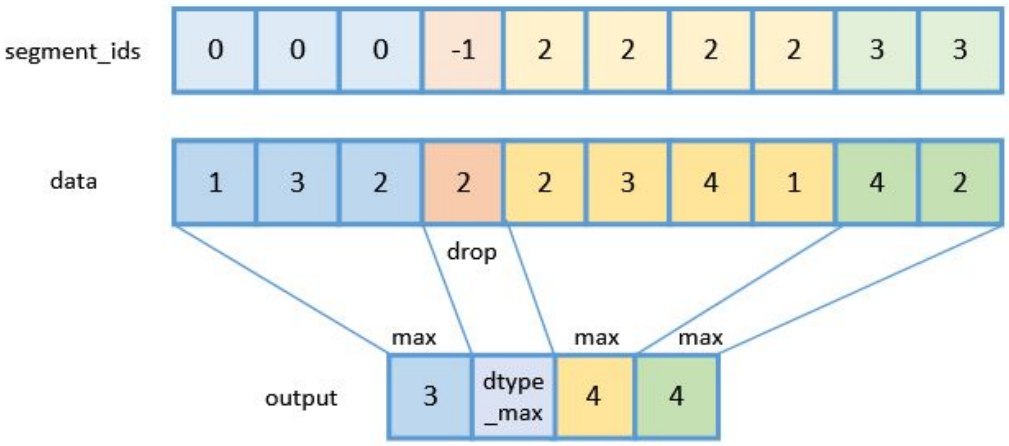
```
import tvm
import te.lang.cce
shape = (5,1024)
input_dtype = "float16"
data = tvm.placeholder(shape, name="data1", dtype=input_dtype)
segment_ids = [1,1,4,5,5]
num_segments = 6
res = te.lang.cce.unsorted_segment_min(data, segment_ids, num_segments)
# res.shape = (6,1024)
# res[0] = 65504(float16的最大值)
# res[1] = min(data[0], data[1])
# res[2] = 65504
# res[3] = 65504
# res[4] = data[2]
# res[5] = min(data[3], data[4])
```

3.33 te.lang.cce.unsorted_segment_max(tensor, segment_ids, num_segments, init_value=0)

使用数组segment_ids对tensor进行分段求最大值。假设输入为data，输出为output，则output[i] = max(data[j...])，其中“j...”是一个数组，“j...”中的元素j满足：segment_ids[j] == i。

如果某个下标i在segment_ids中没有出现,则output[i] = init_value。比如下图中，1在segment_ids中没有出现，则output[1] = 0。

如果segment_ids中某个值为负数，则对应位置的data中的值则丢弃。比如下图中，segment_ids[3] = -1，则data[3]的值被丢弃，不参与计算。



segment_ids其长度必须和data的第一维的长度相同。num_segments必须大于等于segment_ids的最大值加1。

支持的数据类型：float16、float32、int32。

该接口在segment_compute.py中定义。

参数说明

- tensor：输入tensor，必须是如下数据类型之一：float16、float32、int32。
- segment_ids：一维数组，对输入tensor进行分段的数组，其长度必须和输入tensor的第一维的长度相同。同时支持有序和无序。
- num_segments：输出tensor的第一维的长度。其值必须大于等于segment_ids的最大值加1。
- init_value：当segment_ids中某个下标不存在时，其输出的默认值。根据算子的实现来确定。默认值为0。

返回值

res_tensor：表示计算后的tensor。

调用示例

```
import tvm
import te.lang.cce shape = (5,1024)
input_dtype = "float16"
data = tvm.placeholder(shape, name="data1",type=input_dtype)
segment_ids = [1,1,4,5,5]
num_segments = 6
res = te.lang.cce.unsorted_segment_max(data,segment_ids,num_segments)
# res.shape = (6,1024)
# res[0] = 65504(float16的最大值)
# res[1] = max(data[0], data[1])
# res[2] = 65504
# res[3] = 65504
# res[4] = data[2]
# res[5] = max(data[3], data[4])
```

3.34 te.lang.cce.concat(raw_tensors, axis)

在指定轴上对输入的多个Tensor进行重新连接。

输入raw_tensors为多个Tensor，数据类型相同。

如果raw_tensors[i].shape = [D0, D1, ... Daxis(i), ...Dn]，沿着轴axis连接后的结果的shape为：[D0, D1, ... Raxis, ...Dn]。

其中：Raxis = sum(Daxis(i))。

对输入tensor来说，除了轴axis以外，其他轴的维度要完全一致。

例如：

```
t1 = [[1, 2, 3], [4, 5, 6]]
t2 = [[7, 8, 9], [10, 11, 12]]
concat([t1, t2], 0)  # [[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]
concat([t1, t2], 1)  # [[1, 2, 3, 7, 8, 9], [4, 5, 6, 10, 11, 12]]

# tensor t1的shape为 [2, 3]
# tensor t2的shape为 [2, 3]
concat([t1, t2], 0).shape  # [4, 3]
concat([t1, t2], 1).shape  # [2, 6]
```

参数axis也可以为负数，表示从维度的最后开始计算，表示第axis + len(shape)跟轴。

例如：

```
t1 = [[[1, 2], [2, 3]], [[4, 4], [5, 3]]]
t2 = [[[7, 4], [8, 4]], [[2, 10], [15, 11]]]
concat([t1, t2], -1)
```

结果为：

```
[[[ 1,  2,  7,  4],
  [ 2,  3,  8,  4]],

 [[ 4,  4,  2, 10],
  [ 5,  3, 15, 11]]]
```

支持的数据类型：int8、uint8、int16、int32、float16、float32。

该接口在concat_compute.py中定义。

参数说明

- raw_tensors：tensor list，list类型，元素为tvm.tensor，且tensor shape的最后一维要32字节对齐。
- axis：做 concat 操作的轴，取值范围：[-d,d-1]，其中d是raw_tensor的维数。

返回值

res_tensor：重新连接后的tensor，tvm.tensor类型。

调用示例

```
import tvm
import te.lang.cce
shape1 = (64,128)
shape1 = (64,128)
input_dtype = "float16"
data1 = tvm.placeholder(shape1, name="data1", dtype=input_dtype)
data2 = tvm.placeholder(shape2, name="data1", dtype=input_dtype)
data = [data1, data2]
res = te.lang.cce.concat(data, 0)
# res.shape = (128,128)
```

3.35 te.lang.cce.conv(*args)

在给定4-D输入和filter的情况下计算2-D卷积。

要求输入Tensor和filter Tensor的格式均为NCHW。

接口可以支持bias，支持的数据类型float16。

该接口在conv_compute.py中定义。

参数说明

*args：是一个list，参数数量可变。

- bias场景参数列表为（其中hasBias需要为True）：
  A, W, B, res_dtype, padh, padw, strideh, stridew, hasBias
- 非bias场景参数列表为（其中hasBias需要为False）：
  A, W, res_dtype, padh, padw, strideh, stridew, hasBias
  其中：
  - A：输入Tensor，即卷积计算的feature map。
  - W：filter Tensor，即卷积计算的卷积核。
  - B：bias Tensor，即卷积计算的偏置。
  - res_dtype：输出Tensor的数据类型，即卷积计算结果的数据类型。
  - padh：padding的高，即卷积计算在feature map的H方向上的填充数。
  - padw：padding的宽，即卷积计算在feature map的W方向上的填充数。
  - strideh：H方向的步长，即卷积计算filter在feature map的H方向上移动的步长。
  - stridew：W方向的步长，即卷积计算filter在feature map的W方向上移动的步长。
  - hasBias：是否带bias。
- 约束：假设feature map的shape为（Fn, Fc, Fh, Fw），filter的shape为（Wn, Wc, Wh, Ww），卷积结果的输出shape为（On, Oc, Oh, Ow），padh记为Ph，padw记为Pw，strideh记为Sh，stridew记为Sw，上述各参数之间需满足如下关系：
  - Fc = Wc
  - On = Fn
  - Oc = Wn
  - Oh = ((Fh + 2Ph - Wh) / Sh) + 1
  - Ow = ((Fw + 2Pw - Ww) / Sw) + 1

返回值

res_tensor：表示卷积计算的tensor，即卷积计算的结果输出。

调用示例

```
import tvm
import te.lang.cce
```

```
shape_in = (64,64)
shape_w = (3,3)
in_dtype = "float16"
A = tvm.placeholder(shape_in, name='A', dtype=in_dtype)
W = tvm.placeholder(shape_w, name='W', dtype=in_dtype)
b_shape = (shape_w[0], )
B = tvm.placeholder(b_shape, name='B', dtype=res_dtype)
padh = 0
padw = 0
strideh = 1
stridew = 1
res = te.lang.cce.conv(A, W, B, in_dtype, padh, padw, strideh, stridew, True)
# res = A * W + B
```

3.36 te.lang.cce.compute_four2five(input, raw_shape_4D)

把给定4-D “NCHW”数据格式转换为5-D “NC1HWC0”数据格式。支持的数据类型：float16。

该接口在dim_conv.py中定义。

参数说明

- input：输入tensor，4-D格式(N, C, H, W)，tvm.tensor类型。
- raw_shape_4D：输入tensor的维度。

返回值:

res_tensor：转换为5-D格式(N, C1, H, W, C0)后的tensor，tvm.tensor类型

调用示例

```
import tvm
import te.lang.cce
raw_shape = (N,C,H,W)
in_dtype = "float16"
input = tvm.placeholder(raw_shape, name='input', dtype=in_dtype)
res = te.lang.cce.compute_four2five(input, raw_shape)
# res.shape = (N,(C+15)//16,H,W,16)
```

3.37 te.lang.cce.compute_five2four(input, raw_shape_4D)

把给定5-D “NC1HWC0”数据格式转换为4-D “NCHW”数据格式。支持的数据类型：float16。

该接口在dim_conv.py中定义。

参数说明

- input：输入tensor，5-D格式(N, C1, H, W, C0)，tvm.tensor类型。
- raw_shape_4D：转换后tensor的维度。

返回值

res_tensor：转换为4-D格式(N, C, H, W)后的tensor，tvm.tensor类型。

调用示例

```
import tvm
import te.lang.cce
raw_shape = (N,C,H,W)
input_shape = (N,(C+15)//16,H,W,16)
in_dtype = "float16"
input = tvm.placeholder(input_shape, name='input', dtype=in_dtype)
res = te.lang.cce.compute_five2four(input, raw_shape)
# res.shape = (N,C,H,W)
```

3.38 te.lang.cce.matmul(tensor_a, tensor_b, trans_a=False, trans_b=False, alpha_num=1.0, beta_num=0.0, tensor_c=None)

矩阵乘，计算：tensor_c=alpha_num * trans_a(tensor_a) * trans_b(tensor_b) + beta_num * tensor_c。

tensor_a与tensor_b的shape后两维(经过对应转置)需要满足矩阵乘(M, K) * (K, N) = (M, N)，且batch数只支持1。tensor_a数据排布要满足L0A的分形结构，tensor_b要满足L0B的分形结构，mini形态下，数据类型只支持float16。

该接口在mmad_compute.py中定义。

参数说明

- tensor_a：A矩阵，tvm.tensor类型。
- tensor_b：B矩阵，tvm.tensor类型
- trans_a：A矩阵是否转置，bool类型。
- trans_b：B矩阵是否转置，bool类型
- alpha_num : A*B矩阵系数，只支持1.0
- beta_num : C矩阵系数，只支持0.0
- tensor_c : C矩阵，tvm.tensor类型，由于beta_num只支持0.0，此参数为预留扩展接口

返回值

tensor_c：根据关系运算计算后得到的tensor，tvm.tensor类型。

调用示例

```
import tvm
import te.lang.cce
a_shape = (1024, 256)
b_shape = (256, 512)
a_fractal_shape = (a_shape[0] // 16, a_shape[1] // 16, 16, 16)
b_fractal_shape = (b_shape[0] // 16, b_shape[1] // 16, 16, 16)
in_dtype = "float16"
tensor_a = tvm.placeholder(a_fractal_shape, name='tensor_a', dtype=in_dtype)
```

```
tensor_b = tvm.placeholder(a_fractal_shape, name='tensor_b', dtype=in_dtype)
res = te.lang.cce.matmul(tensor_a, tensor_b, False, False)
```

4 build 接口

build接口的作用是把定义好的计算过程生成schedule对象，并进行build生成cce算子文件。

4.1 topi.generic.auto_schedule(outs)

生成dsl的schedule。该接口在cce.py中定义。

参数说明

outs：对算子的计算图描述，就是DSL。

返回值:

schedule：算子的计算schedule。

调用示例

```
import te.lang.cce
from te import tvm
import topi.generic

shape = (28,28)
dtype = "float16"
# 定义输入
data = tvm.placeholder(shape, name="data", dtype=dtype)
# 描述算子计算过程
res = te.lang.cce.vabs(data)
with tvm.target.cce():
    # 生成schedule对象
     sch = topi.generic.auto_schedule(res)
```

4.2 te.lang.cce.cce_build_code(sch, config_map = {})

对schedule打印lower code或者进行build。该接口在cce_schedule.py中定义。

参数说明

- sch：tvm.schedule，schedule to build or to print lower code。
- config_map：build的参数配置，是一个字典，默认是{}并使用默认配置，包含如下key。
  - print_ir：是否打印lower IR code，默认是True。
  - need_build：是否进行build，默认是True。
  - name：算子的名字，默认是`cce_op`。
  - tensor_list：算子的输入和输出tensor列表，输入是placeholder接口返回的tensor对象，输出是经过计算后的tensor对象，必填值，否则会报错。而且这个列表决定了生成算子的kernel函数的参数的顺序，和此list中的输入和输出的顺序是一致的。

返回值

无。

调用示例

```
import te.lang.cce
from te import tvm
from topi import generic
# 定义输入占位符
data = tvm.placeholder(shape, name="data", dtype=dtype)
with tvm.target.cce():
    # 描述算子计算过程
    res = te.lang.cce.vabs(data)
    # 生成schedule对象
    sch = generic.auto_schedule(res)
# 定义build配置参数
config = {"print_ir" : True,
        "need_build" : True,
        "name" : "abs_28_28_float16",
        "tensor_list" : [data,res]
    }
# build算子
te.lang.cce.cce_build_code(sch, config)
```

5 融合接口

TE算子融合功能需要对单算子的融合能力和入参等信息做预处理，因此在生成单算子时需要调用指定API，如下对该API的使用方法和限制做详细介绍。

5.1 bool BuildTeCustomOp(std::string ddkVer, std::string opName, std::string opPath, std::string opFuncName, const char *format, ...)

调用该接口用于生成单算子的*.o文件和*.json文件。您可以在DDK包的安装目录下的“ddk/include/inc/custom/custom_op.h”文件中查看接口的定义。

参数说明

- ddkVer：指产品的版本号。

  版本号统一规则：x.y.z.patch.B***。

  x：对应VR版本；

  y：对应C版本；

  z：对应发布计数；

  path：补丁号，可选参数；

  B***：内部B版本号；

  具体版本号参考DDK包的安装目录下ddk_info文件中的配置。
- opName：指网络中单个算子节点名称，其值不能为空，且名称中只能包含字母、数字、下划线或中划线。
- opPath：指算子文件的路径，以“/”区分目录且文件名称不带扩展名，其值不能为空。文件名称中只能包含字母、数字、下划线或中划线。
- opFuncName：指算子文件中算子函数名称。函数名称中只能包含字母、数字、下划线或中划线。
- format：算子的入参，属于变长变量，需要与算子的参数顺序保持一致。

返回值:

- true：表示生成单算子成功
- false：表示生成单算子失败。

调用示例

```
std::string FilePath   = "topi/cce/caffe_reduction_layer";
std::string FuncName   = "caffe_reduction_layer_cce";
std::string KernelName = "cce_reductionLayer_5_10_5_5_float16__1_SUMSQ_1_0";

// i => int; s => string; f => dobule; O => bool, and bool value is Py_True or Py_False
te::BuildTeCustomOp(1.1.1.patch.B001, "reduction", FilePath, FuncName,"(i,i,i,i), s, i, s, f, s", 5, 5, 5, 5,
"float16", 1, "SUM", 1.0,KernelName.c_str());
```

6 编译依赖接口

如下接口只在算子编译过程中用到，用户在编写算子时不会直接调用。您可以在DDK包的安装目录下的“ddk/include/inc/tensor_engine/cce_aicpu_3rd_party.h”文件中查看接口的定义。

- __aicpu__ void *aicpu_malloc(unsigned int size)
  该接口用于实现内存申请，入参size表示申请内存的大小，返回申请内存的指针。
- __aicpu__ void aicpu_free(void * ptr)
  该接口用于实现内存释放，ptr为内存指针。
- __aicpu__ double log(double x)
  该接口用于实现对x取log值，double类型。
- __aicpu__ double exp(double x)
  该接口用于实现对x取exp值，double类型。
- __aicpu__ double round(double x)
  该接口用于实现对x四舍五入，double类型。
- __aicpu__ double floor(double x)
  该接口用于实现对x向下取整，double类型。
- __aicpu__ double ceil(double x);
  该接口用于实现对x向上取整，double类型。
- __aicpu__ double trunc(double x);
  该接口用于实现对x截断，double类型。
- __aicpu__ double sqrt(double x);
  该接口用于实现对x开平方，double类型。
- __aicpu__ float logf(float x)
  该接口用于实现对x取log值，double类型。
- __aicpu__ float expf(float x)
  该接口用于实现对x取exp值，float类型。
- __aicpu__ float roundf(float x)
  该接口用于实现对x四舍五入，float类型。
- __aicpu__ float floorf(float x)

该接口用于实现对x向下取整，float类型。

- __aicpu__ float ceilf(float x);
  该接口用于实现对x向上取整，float类型。
- __aicpu__ float truncf(float x);
  该接口用于实现对x截断，float类型。
- __aicpu__ float sqrtf(float x);
  该接口用于实现对x开平方，float类型。

7 使用方式

7.1 使用示例

下面通过一个简单的示例，将上面的接口串起来。该示例是实现一个简单的支持float16数据类型的求绝对值的算子。

```
import te.lang.cce
from te import tvm
import topi
shape = (28,28)
dtype = "float16"
# 定义输入
data = tvm.placeholder(shape, name="data", dtype=dtype)

# 描述算子计算过程
res = te.lang.cce.vabs(data)

with tvm.target.cce():
    # 生成schedule对象
     sch = topi.generic.auto_schedule(res)

# 定义build配置参数
config = {"print_ir" : True,
          "need_build" : True,
          "name" : "abs_28_28_float16",
          "tensor_list" : [data,res]}
# build算子
te.lang.cce.cce_build_code(sch, config)
```

7.2 异常处理

接口如果执行异常，一般都是由于错误的入参引起的。下边例子给出了tensor_list不全给出的错误信息。

代码片段

```
data = tvm.placeholder(shape, name="data", dtype=inp_dtype)
with tvm.target.cce():
   res = te.lang.cce.vabs(data)
```

```
    sch = generic.auto_schedule(res)
    config = {"print_ir": need_print,
    "need_build": need_build,
    "name": kernel_name,
    "tensor_list": [res]}
te.lang.cce.cce_build_code(sch, config)
```

执行会发生如下错误：

```
Traceback (most recent call last):
  File "llt/tensor_engine/ut/testcase_python/tf_abs/test_tf_abs_cce.py", line 71, in test_cce_tf_abs_99991_fp16
    tf_abs_cce((99991,), dtype = "Float16", need_build = False, need_print = False, kernel_name =
"cce_tf_abs")
  File "/home1/repotvm/tensor_engine/topi/python/topi/cce/tf_abs.py", line 68, in tf_abs_cce
    te.lang.cce.cce_build_code(sch, config)
  File "/home1/repotvm/tensor_engine/python/te/lang/cce/te_schedule/cce_schedule.py", line 381, in
cce_build_code
    _build(sch, tensor_list, local_config_map["name"])
  File "/home1/repotvm/tensor_engine/python/te/lang/cce/te_schedule/cce_schedule.py", line 338, in _build
    mod = tvm.build(sch, tensor_list, device, name=name)
  File "/home1/repotvm/tensor_engine/python/te/tvm/build_module.py", line 432, in build
    binds=binds)
  File "/home1/repotvm/tensor_engine/python/te/tvm/build_module.py", line 353, in lower
    stmt = ir_pass.StorageFlatten(stmt, binds, 64)
  File "/home1/repotvm/tensor_engine/python/te/tvm/_ffi/function.py", line 280, in my_api_func
    return flocal(*args)
  File "/home1/repotvm/tensor_engine/python/te/tvm/_ffi/_ctypes/function.py", line 183, in __call__
    ctypes.byref(ret_val), ctypes.byref(ret_tcode)))
  File "/home1/repotvm/tensor_engine/python/te/tvm/_ffi/base.py", line 66, in check_call
    raise TVMError(py_str(_LIB.TVMGetLastError()))
TVMError: [17:12:02] /home1/repotvm/tensor_engine/src/pass/storage_flatten.cc:249: Check failed: it !=
buf_map_.end() Cannot find allocated buffer for placeholder(data, 0x27d7290)
```

参数更正为如下情况后，问题得到解决。

```
"tensor_list" : [data, res]
```

8 附录

8.1 修订记录

文档版本	发布日期	修改说明
01	2020-05-09	第一次正式发布版本。