寒武纪 BANGPy 开发者手册

版本 *1.1.0*

2021 年 3 月 31 日

Cambricon 寒武纪

目录

Cambricon 寒武纪

插图目录

Cambricon 寒武纪

表格目录

1 版权声明

免责声明

中科寒武纪科技股份有限公司（下称“寒武纪”）不代表、担保（明示、暗示或法定的）或保证本文件所含信息，并明示放弃对可销售性、所有权、不侵犯知识产权或特定目的适用性做出任何和所有暗示担保，且寒武纪不承担因应用或使用任何产品或服务而产生的任何责任。寒武纪不应对因下列原因产生的任何违约、损害赔偿、成本或问题承担任何责任：（1）使用寒武纪产品的任何方式违背本指南；或（2）客户产品设计。

责任限制

在任何情况下，寒武纪都不对因使用或无法使用本指南而导致的任何损害（包括但不限于利润损失、业务中断和信息损失等损害）承担责任，即便寒武纪已被告知可能遭受该等损害。尽管客户可能因任何理由遭受任何损害，根据寒武纪的产品销售条款与条件，寒武纪为本指南所述产品对客户承担的总共和累计责任应受到限制。

信息准确性

本文件提供的信息属于寒武纪所有，且寒武纪保留不经通知随时对本文件信息或对任何产品和服务做出任何更改的权利。本指南所含信息和本指南所引用寒武纪文档的所有其他信息均“按原样”提供。寒武纪不担保信息、文本、图案、链接或本指南内所含其他项目的准确性或完整性。寒武纪可不经通知随时对本指南或本指南所述产品做出更改，但不承诺更新本指南。

本指南列出的性能测试和等级要使用特定芯片或计算机系统或组件来测量。经该等测试，本指南所示结果反映了寒武纪产品的大概性能。系统硬件或软件设计或配置的任何不同会影响实际性能。如上所述，寒武纪不代表、担保或保证本指南所述产品将适用于任何特定用途。寒武纪不代表或担保测试每种产品的所有参数。客户全权承担确保产品适合并适用于客户计划的应用以及对应用程序进行必要测试的责任，以期避免应用程序或产品的默认情况。

客户产品设计的脆弱性会影响寒武纪产品的质量和可靠性并导致超出本指南范围的额外或不同的情况和/或要求。

知识产权通知

寒武纪和寒武纪的标志是中科寒武纪科技股份有限公司在中国和其他国家的商标和/或注册商标。其他公司和产品名称应为与其关联的各自公司的商标。

式复制、重制、修改、出版、上传、发布、传输或分发本指南。除了客户使用本指南信息和产品的权利，根据本指南，寒武纪不授予其他任何明示或暗示的权利或许可。未免疑义，寒武纪不根据任何专利、版权、商标、商业秘密或任何其他寒武纪的知识产权或所有权对客户授予任何（明示或暗示的）权利或许可。

- 版权声明
- © 2021 中科寒武纪科技股份有限公司保留一切权利。

2 前言

2.1 版本记录

表 2.1: 版本记录

文档名称	寒武纪 BANGPy 开发者手册
版本号	V 1.1.0
作者	Cambricon
修改日期	2021 年 3 月 31 日

2.2 更新历史

- **V1.1.0**

  更新时间: 2021 年 3 月 31 日

  更新内容:

  - 增加了 BANGPy 对多架构的适配支持，如 MLU370。
  - 修改数据搬运算子 接口，新增 `dst_cluster` 参数。
  - 支持多编程模型的代码生成。

- **V1.0.0**

  更新时间: 2021 年 2 月 10 日

  更新内容:

  - 扩充了 TCP 的算子，增强通用性。
  - 增加了 BANGPy 对多架构的适配支持，如 MLU290。
  - TCP 增加对可变参数的支持，支持算子输入输出张量形状可变。
  - TCP 强化算子参数检查，增加函数调用栈追踪，运行错误直接定位到算子源代码。
  - TCP 新增流水线优化与存储复用优化功能。
  - 增加 TensorOp 接口介绍。
  - 增加 TensorOp API 列表。

- **V0.9.0**

  更新时间: 2020 年 12 月 05 日

  更新内容:

  – 修改命名，规范化接口。
  – 合并 cycle 类算子，降低算子复杂度，提高易用性。
  – 增加张量方法，如 reshape，reinterpret_cast 等。
  – 新增标量支持。
  – 简化 memcpy 接口，使用张量参数，去除 stride 等参数，memcpy 接口直接使用张量编程。
  – 增加数值类型，将 string 表达的数值类型修改为 BANGPy 的数值类型，如将”int8”改为 bangpy.int8。
  – 增加张量索引方式，支持使用标量或者大小为 1 的张量索引一个张量。
  – 增加 any/all 接口，控制流语句可使用 any/all 接口支持多重判断。
  – 新增控制流接口。

- **V0.8.0**

  更新时间: 2020 年 11 月 05 日

  更新内容:

  – 初始版本。

3 概述

随着神经网络技术的广泛应用，网络模型在专用硬件平台上的快速部署成为强需求。BANGPy 是一种基于 Python 的编程框架，用于神经网络算子开发与网络搭建，为寒武纪 MLU 系列硬件平台提供便捷的软件接口。

BANGPy 的优点包括:

1. 面向张量编程:
   BANGPy 提供了张量（tensor）声明 API，并以张量作为基本操作对象。
2. 多种输出形式:
   BANGPy 提供了算子 BANG C 源代码和库文件两种输出形式，方便用户二次开发。
3. 简化面向 BANG 架构的编程:
   BANGPy 对 BANG 架构进行了更高层次的抽象与封装，相比于 BANG C 极大地减轻用户编写算子的工作量，从而提高了算子开发效率。
4. 运行时支持:
   BANGPy 可以直接调用 MLU 硬件资源，进行算子开发验证。

开发前必读 章节全面介绍了 BANGPy 软件栈结构、MLU 硬件架构、软件环境配置和安装。

基本概念 章节提供了本手册所用到的神经网络算子相关知识和术语解释。

算子开发 章节通过具体实例，详细说明了算子开发的代码实现、编译、验证等流程。

TCP 接口 章节罗列了 BANGPy 所有算子开发底层接口（TCP 接口）。

TensorOp 接口 章节罗列了 BANGPy 所有算子开发高层接口（TensorOp 接口）。

通用接口 章节罗列了 BANGPy 所有通用 API。

4 开发前必读

4.1 软件栈结构

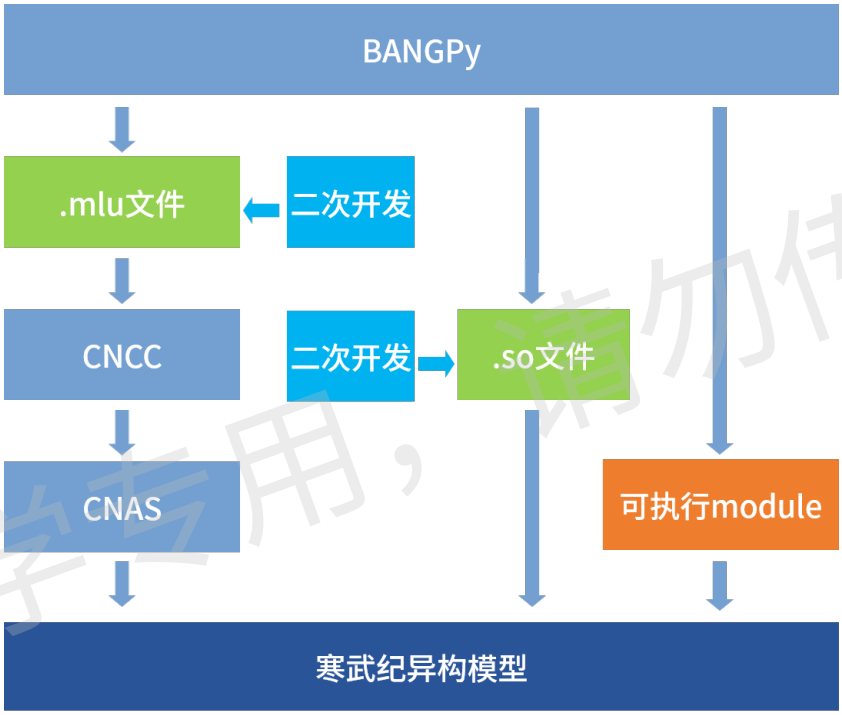

图 4.1: BANGPy 软件栈示意图

如图，使用 BANGPy 接口描述的算子实现，经过 BANGPy 编译优化之后，会得到对应的 MLU 端 BANG C 代码（.mlu 为后缀的文件）、CPU 端动态链接库文件（.so 为后缀的文件）、可执行 module（Python 文件中的变量）。

BANGPy 编译生成的算子有两种格式：

- BANG C 代码：用户可以基于 BANG C 源码修改，进行二次开发。
  - CNCC(Cambricon Neuware Compiler Collection，寒武纪 BANG C 语言编译器) 将 BANG C 代码编译为 MLISA 语言。
  - 通过 CNAS(Cambricon Neuware Assembler，寒武纪 MLISA 语言编译器) 生成二进制文件。
- 动态链接库：以库的方式提供算子，供用户二次开发使用。

BANGPy 能够编译算子，生成可执行 module，用来验证算子的正确性。

关于寒武纪异构模型请参见《寒武纪 CNRT 用户手册》**异构编程**章节的描述。

4.2 硬件架构

MLU270 是主要应用于云端的一款产品，结构如图 4.2 所示。其芯片由 4 个 Cluster 组成，每个 Cluster 由 4 个计算内核组成。

每个计算内核包含片上存储 NRAM(Neural-RAM, 共 512KB) 和 WRAM(Weight-RAM, 共 1MB)，以及计算单元 NFUs(Neural Function Units)，每个核有独立映射的 LDRAM(Local-DRAM)，所有核共享片外存储 GDRAM(Global-DRAM)。

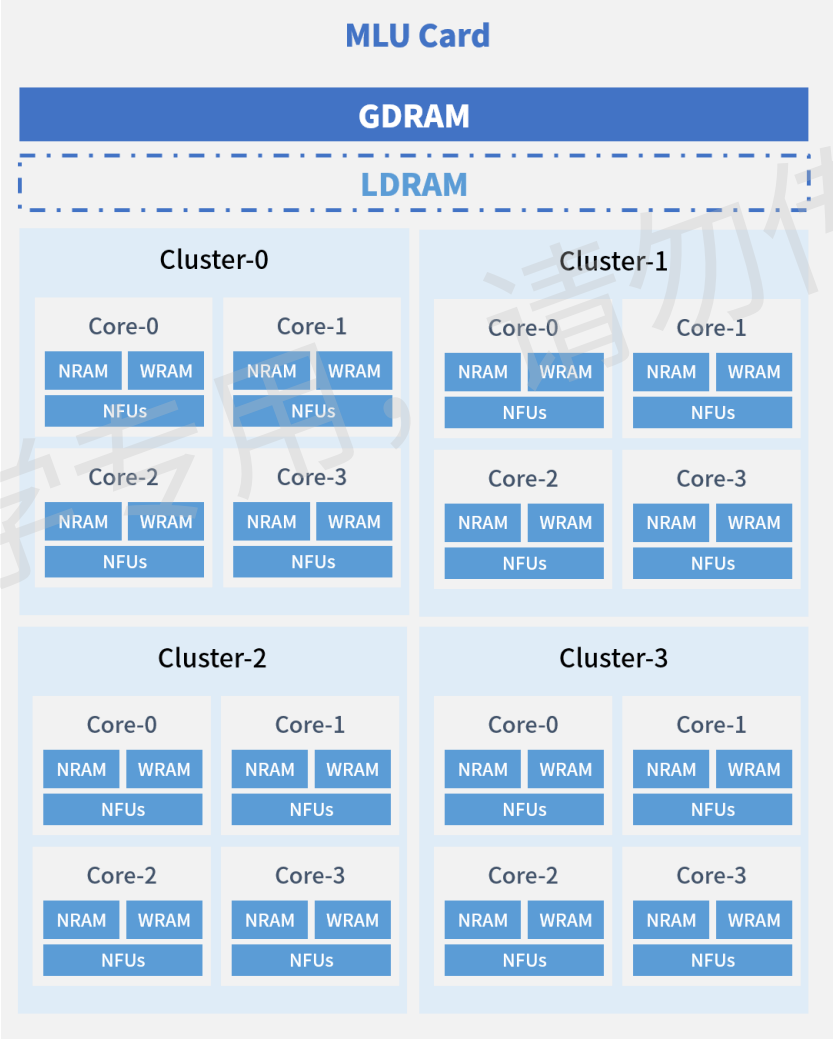

图 4.2: MLU270 结构示意图

在 BANGPy 中，MLU 硬件上运行的程序被称为 Kernel。用户可以通过设置 task_num，task_type 对任务进行自动多核并行展开。task_num 可设置为 1、4、8 或 16，对应 task_type 分别设置为 `TaskType.BLOCK`

、TaskType.UNION1 、TaskType.UNION2 或 TaskType.UNION4 ，表示同时启用 1 个、4 个、8 个或 16 个内核执行任务。

在 BANGPy 中，用户可以调用接口:

```
with for_range(0, task_num, task_num=task_num, task_type=task_type) as task_id:
    ...
```

来进行多核任务展开，并获得只读内置变量 taskId。其中 taskId 的取值范围为 0～(task_num-1)，代表和索引不同核。

用户在定义张量时，可以设置 scope=global/nram/wram，决定变量所在地址空间。

4.3 环境依赖

- 操作系统：目前 BANGPy 仅支持 Ubuntu 16.04 操作系统。
- 软件包依赖：
  - CNToolkit 软件包，请使用 CNToolkit 1.7.0 及以上版本。
  - LLVM（Low Level Virtual Machine，底层虚拟机）请使用 llvm- 6.0 版本。
  - Python 包，请使用 Python3.5，对应包的版本有 numpy==1.18.0 scipy==1.1.0 setuptools==41.6.0，decorator，attrs，psutil。

4.4 软件安装

4.4.1 环境安装

1. CNToolkit 软件包的安装步骤参见《寒武纪 CNToolkit 软件包安装升级使用手册》。
2. LLVM 安装步骤如下：

   ```
   apt-get install llvm-6.0
   ```

   运行：

   ```
   llvm-config-6.0
   ```

   或者：

   ```
   llvm-config
   ```

3. 如下回显信息，表示已经成功安装 LLVM 软件包。

   ```
   usage: llvm-config <OPTION>... [<COMPONENT>...]
   ```



```
Get various configuration information needed to compile programs which use
LLVM.  Typically called from 'configure' scripts.  Examples:
  llvm-config --cxxflags
  llvm-config --ldflags
  llvm-config --libs engine bcreader scalaropts

Options:
  --version         Print LLVM version.
  --prefix          Print the installation prefix.
  --src-root        Print the source root LLVM was built from.
  --obj-root        Print the object root used to build LLVM.
  --bindir          Directory containing LLVM executables.
  --includedir      Directory containing LLVM headers.
  --libdir          Directory containing LLVM libraries.
  --cmakedir        Directory containing LLVM cmake modules.
  --cppflags        C preprocessor flags for files that include LLVM headers.
  --cflags          C compiler flags for files that include LLVM headers.
  --cxxflags        C++ compiler flags for files that include LLVM headers.
  --ldflags         Print Linker flags.
  --system-libs     System Libraries needed to link against LLVM components.
  --libs            Libraries needed to link against LLVM components.
  --libnames        Bare library names for in-tree builds.
  --libfiles        Fully qualified library filenames for makefile depends.
  --components      List of all possible components.
  --targets-built   List of all targets currently built.
  --host-target     Target triple used to configure LLVM.
  --build-mode      Print build mode of LLVM tree (e.g. Debug or Release).
  --assertion-mode  Print assertion mode of LLVM tree (ON or OFF).
  --build-system    Print the build system used to build LLVM (always cmake).
  --has-rtti        Print whether or not LLVM was built with rtti (YES or NO).
  --has-global-isel Print whether or not LLVM was built with global-isel support (ON or OFF).
  --shared-mode     Print how the provided components can be collectively linked (`shared`␣
↪or `static`).
  --link-shared     Link the components as shared libraries.
  --link-static     Link the component libraries statically.
  --ignore-libllvm  Ignore libLLVM and link component libraries instead.
Typical components:
  all               All LLVM libraries (default).
  engine            Either a native JIT or a bitcode interpreter.
```

3. 安装必须的 Python 包：


```
pip install numpy==1.18.0 scipy==1.1.0 setuptools==41.6.0 decorator attrs psutil
```

4.4.2 BANGPy 安装

注意:

安装 BANGPy 前，请确认已安装相关环境依赖的软件包，安装步骤请参见环境安装 。

1. 使用如下命令安装 BANGPy：

```
pip install bangpy-1.1.0-py3-none-any.whl
```

2. BANGPy 安装完毕之后，算子开发样例文件在如下路径：

```
${path_to_python_package}/site-packages/bangpy/samples
```

其中 ${path_to_python_package} 是用户安装 Python 包的路径。

3. 运行如下命令，运行样例文件：

```
python ${path_to_python_package}/site-packages/bangpy/samples/tcp/test_topk.py
```

回显信息如下所示，表明样例运行成功。

```
CNRT: 4.11.0 9287896
topk : 0.813661 ms
topk : 0.793933 ms
topk : 0.797286 ms
topk : 0.216480 ms
topk : 0.214754 ms
topk : 0.233685 ms
topk : 0.189804 ms
topk : 0.182643 ms
topk : 0.145189 ms
topk : 0.150409 ms
topk : 0.141444 ms
topk : 0.151613 ms
topk : 0.189974 ms
topk : 0.186100 ms1
```

在后续的样例或测试运行中，如果程序正常结束，无错误信息提示，则表明算子编译成功，同时，在 MLU 上的运行的结果与 CPU 上的运行结果误差符合预期。

Cambricon 寒武纪

5 基本概念

5.1 算子开发

算子 (operator)，简称 op，表示从一个函数空间到另一个函数空间的映射，在深度学习领域通常对应于神经网络模型中的某个网络层，本手册中算子特指深度学习算法中的运算或操作。

5.1.1 Cambricon Tensor Compute Primitive

BANGPy 提供了一系列 API 用于算子开发，称为 CAMBRICON TCP™（Cambricon Tensor Compute Primitive，寒武纪张量计算原语），以下简称 TCP。详细的 TCP 接口列表请参见TCP 接口 章节。

算子与 TCP 关系：

- 算子可以由单个 TCP 接口直接表达（例如 ReLU 算子）；
- 算子也可以通过多个 TCP 接口组合而成（例如 softmax 算子由 exp、div 等 TCP 接口组成）。具体算子的开发流程，请参见算子开发 章节。

5.1.2 Cambricon TensorOp Interface

BANGPy 还提供了一系列类 numpy 的高层 API 用于算子开发，称为 Cambricon TensorOp Interface（寒武纪张量操作接口），以下简称 TensorOp。

算子与 TensorOp 的关系：

- 算子可以由单个 TensorOp 接口直接表达（例如 ReLU 算子）；
- 算子也可以通过多个 TensorOp 接口组合而成（例如 atanh 算子由 log、div 等 TensorOp 接口组成）。具体算子的开发流程，请参见算子开发 章节。

TensorOp 与 TCP 的不同与联系：

- TCP 提供丰富的底层计算原语，由用户自由管理数据在片上的存储和搬运，编写高性能算子。
- TensorOp 提供高度封装的高层接口，用户不需要管理数据在片上的存储和搬运，快速编写算子。
- 用户在使用 TensorOp 接口编写算子时，还可以在其中嵌入用 TCP 接口所编写的算子，BANGPy 会对这种混合方式编写的算子进行统一编译。具体使用可以参看：samples/mixed_op/test_mix.py 示例 demo。

5.2 数值类型

BANGPy 现支持的 **数值类型 (dtype)** 包括 int8、int16、int32、uint16、uint32、float16 和 float32，分别使用 bangpy.int8、bangpy.int16、bangpy.int32、bangpy.uint16、bangpy.uint32、bangpy.float16 和 bangpy.float32 表示。

5.3 张量

张量 (tensor) 是神经网络中的基本数据单元，数学上表示为多维数组。为了方便起见，本手册将一维数组（向量）、二维数组（矩阵）也称为张量。

在 TCP 中，一个张量由它的 **形状 (shape)** 、**数值类型 (dtype)** 、**名字 (name)** 和 **地址空间 (scope)** 确定。可以用以下方式定义：

```
bp = bangpy.tcp.TCP()
tensor = bp.Tensor(shape, dtype, name, scope)
```

在 TensorOp 中，一个张量由它的 **形状 (shape)** 、**数值类型 (dtype)** 和 **名字 (name)** 确定。可以用以下方式定义：

```
from bangpy import tensor_op as tsop
tensor = tsop.tensor(shape, dtype, name)
```

5.3.1 形状

5.3.1.1 维度和轴

张量的形状描述了多维数组各个 **维度（dim）** 的大小。形状以 tuple of int 的类型作为 TCP 和 TensorOp 的张量创建接口的 shape 参数。

维度又称 **轴 (axis)** ，axis=0 表示张量最高（外）维，axis=1 表示张量从外数第二维，axis 的数值表示的维度以此类推。

5.3.1.2 数据排布

数据排布 (layout) 表示数据的排布方式。在神经网络的某些应用场景中，张量的各个维度有着实际含义。以图片处理的卷积神经网络为例，张量通常是四维，对应 N, H, W, C。其中 N 表示图片张数，H 和 W 分别代表图片的长和宽，C 代表通道数，即，每一张图片上的每一个像素点，都用一个长度为 C 的向量描述。

需要注意的是，在许多神经网络模型中，数据排布为 NCHW，而 MLU 硬件处理的数据排布为 NHWC。所以用户可能需要调用 TCP 中的数据排布转换接口，将 layout 转换成 NHWC，再进行下一步处理。比

如当卷积算子的排布方式为 NCHW，用户可以用转置算子或者间隔取数的方式，将卷积的数据转换成 NHWC，待计算结束后再转回 NCHW。

5.3.2 数值类型

张量的 **数值类型 (dtype)** 表示的是张量中每个标量元素的数值类型，需要用户在创建张量时给定，具体请参见数值类型 节。关于不同 TCP 张量计算接口支持的数值类型，请参见TCP 接口 章节。

5.3.3 面向张量的操作

在 TCP 中，用户可以从一个张量中取出小张量，类似 numpy 的 ndarray 中的 slice 操作，例如：

```
tensor[1]
tensor[1:]
tensor[0:2]
tensor[1, 2]
tensor[1][0]
tensor[1:2][0]
tensor[1, 2, :]
tensor[1, 2][1]
```

用户可以将它作为一个普通的张量使用，也可以直接对其中的某个元素进行赋值，例如：

```
tensor[0][1] = rhs
```

其中 rhs 限制为大小为 1 的 Tensor 类型变量，int/float 类型 Python 变量, 以及由循环变量、BANGPy 内置变量 taskId 构成的表达式。

除此之外，用户可以直接获得张量的如下属性：

```
tensor.dtype # 张量的元素数值类型
tensor.name # 张量在创建时赋予的名字
tensor.scope # 张量所在地址空间
tensor.ndim # 张量的维度数
tensor.shape # 张量的形状
tensor.size # 张量的大小
tensor.is_contiguous # 张量是否连续存储
```

以及对张量进行 reshape 操作：

```
new_tensor = tensor.reshape(new_shape)
```

更多操作请参见TCP 接口 章节。

5.4 标量

在 TCP 中，标量是神经网络的一种数据单元，在数学上表示为一个单独的数值。它对应的是存储在芯片寄存器里的数值。在 TCP 中，一个标量由它的 **数据类型 (dtype)** ，**名称 (name)** 和 **初始值 (value)** 确定，可以用以下方式定义：

```
bp = bangpy.tcp.TCP()
scalar = bp.Scalar(dtyp, name, value)
```

5.4.1 数值类型

标量的 **数值类型 (dtype)** 表示的是标量元素的数值类型。具体可参见数值类型 节。在定义标量时可给出初始值，若用户在定义时未给定初始值时，表示声明一个标量，该标量没有初始值。

6 算子开发

6.1 TCP 算子开发流程

本节以张量加算子为例，介绍算子开发流程和常见接口使用。

图 6.1: TCP 算子开发流程示意图

图 6.1 展示了 TCP 算子开发的大致流程，主要步骤如下：

1. 创建 TCP 容器。
2. 计算描述：在 TCP 容器中使用 TCP 接口描述计算，主要包括定义张量、张量搬运、张量计算、并行编程以及条件和循环控制等。
3. 编译：调用编译接口生成可执行 module，该可执行 module 可以保存成 CPU 端的 host.so 文件和 MLU 端的 device.mlu 文件。
4. 运行验证：将输入、初始化的输出数据传入可执行 module 中运行，检查算子的正确性。
5. 调试分析：针对运行结果，进行调试分析。

6.1.1 样例

实现两个张量相加的部分代码如下，完整代码参见安装目录/tests/tcp/test_get_started.py：

```
...
# 获取 numpy 的输入输出数据
data_in0 = np.random.uniform(low=-10, high=10, size=SHAPE)
data_in1 = np.random.uniform(low=-10, high=10, size=SHAPE)
data_out = data_in0 + data_in1

length = np.prod(data_in0.shape)

# 创建 TCP 容器
bp = tcp.TCP()

# 计算分块次数
assert length % task_num == 0
core_wl = 3 * length // task_num
loop_num = np.ceil(core_wl * dtype_sz / NRAM_SIZE)
core_wl //= 3
while core_wl % loop_num != 0:
    loop_num += 1
loop_wl = int(core_wl // loop_num)

# 定义输入输出张量
tensor_in0 = bp.Tensor(shape=data_in0.shape, name="INPUT0", dtype=dtype, scope="global")
tensor_in1 = bp.Tensor(shape=data_in1.shape, name="INPUT1", dtype=dtype, scope="global")
tensor_out = bp.Tensor(shape=data_out.shape, name="OUTPUT", dtype=dtype, scope="global")

# 描述多核并行逻辑和分块计算逻辑
task_type = TaskType.get_task_type(task_num)
with bp.for_range(0, task_num, task_num=task_num, task_type=task_type) as task_id:

    # 定义 NRAM 上的中间张量
    tensor_in0_n = bp.Tensor(shape=(loop_wl,), name="INPUT0_N", dtype=dtype, scope="nram")
    tensor_in1_n = bp.Tensor(shape=(loop_wl,), name="INPUT1_N", dtype=dtype, scope="nram")
    tensor_out_n = bp.Tensor(shape=(loop_wl,), name="OUTPUT_N", dtype=dtype, scope="nram")

    # 描述分块逻辑
    with bp.for_range(0, loop_num) as i:
        start = task_id * core_wl + i * loop_wl
        stop = start + loop_wl
```



```
        # 调用 TCP 接口进行张量搬运与计算
        bp.memcpy(tensor_in0_n, tensor_in0[start:stop])
        bp.memcpy(tensor_in1_n, tensor_in1[start:stop])
        bp.add(tensor_out_n, tensor_in0_n, tensor_in1_n)
        bp.memcpy(tensor_out[start:stop], tensor_out_n)


# 编译生成可执行 module
fvec_add = bp.BuildBANG(inputs=[tensor_in0, tensor_in1], outputs=[tensor_out], kernel_name=
↪"fvec_add")

# 设备端空间申请与张量搬运
ctx = bangpy.context(0)
data_in0_dev = bangpy.Array(data_in0.astype(dtype), ctx)
data_in1_dev = bangpy.Array(data_in1.astype(dtype), ctx)
data_out_dev = bangpy.Array(np.zeros(data_out.shape, dtype), ctx)

# 准备一个空目录 dirname，用来保存 BANG C 代码和.so 文件
fvec_add.save(dirname)

# 直接启用设备运行，验证结果
fvec_add(data_in0_dev, data_in1_dev, data_out_dev)
bangpy.assert_allclose(data_out_dev.asnumpy(), data_out.astype(dtype), rtol=1e-2, atol=1e-2)

# 载入 module，启用设备运行，验证结果
fadd = load_module(dirname, target)
fadd(data_in0_dev, data_in1_dev, data_out_dev)
bangpy.assert_allclose(data_out_dev.asnumpy(), data_out.astype(dtype), rtol=1e-2, atol=1e-2)
```

6.1.2 代码解析

6.1.2.1 创建 TCP 容器

TCP 容器用于创建和操作算子。使用如下命令创建一个 TCP 容器。target 为字符串，代表该 TCP 容器适用的架构类型。当前支持的架构有：MLU270，MLU290``及``MLU370。当未指定 target 时，默认的 target 为 MLU270。

```
from bangpy import tcp
bp = tcp.TCP(target)
```

6.1.2.2 计算描述

在 TCP 容器中使用 TCP 接口描述计算，包括以下流程。

定义张量

定义张量需要给定张量的数值类型、形状、名字、地址空间。算子开发过程中，算子的输入张量和输出张量的地址空间需要定义为 global，例如：

```
tensor_in0 = bp.Tensor(shape=(64000,), name="INPUT0", dtype=bangpy.float16, scope="global")
```

上述示例定义了一个有 64000 个元素的一维张量。TCP 支持动态形状的输入输出张量，用户可以使用 Var 类型在计算描述阶段来描述输入输出张量的形状，TCP 在编译阶段会自动将形状作为算子的输入参数代入算子。例如：

```
shape_in = bp.Var("shape_in")
tensor_in0 = bp.Tensor(shape=(shape_in,), name="INPUT0", dtype=bangpy.float16, scope="global")
```

张量搬运

TCP 计算接口对操作数张量的地址空间有限制，算子输入输出张量的地址空间必须是 global，因此需要对张量进行搬运。例如：

```
bp.memcpy(tensor_in0_n, tensor_in0[start:stop])
```

表示将 scope 为 global 的张量 tensor_in0 中 start 和 stop 之间的数据搬运到 scope 为 NRAM 的张量 tensor_in0_n 中。

关于张量搬运接口的详细描述，请参见TCP 接口 章节。

张量计算

TCP 提供了众多的 API 接口进行张量计算，例如张量加计算：

```
bp.add(tensor_out_n, tensor_in0_n, tensor_in1_n)
```

关于张量计算接口的详细描述，请参见TCP 接口 章节。

定义可变参数

TCP 提供了 Var 接口来定义算子中的可变参数，例如：

```
length = bp.Var("length")
```

并行编程

用户可以使用如下方式实现 task_num 数量的多核并行编程：

```
task_type = TaskType.get_task_type(task_num)
with bp.for_range(0, task_num, task_num=task_num, task_type=task_type) as task_id:
    # for_body
```

条件和循环控制

TCP 提供了对循环和条件判断的支持，例如：

```
with bp.for_range(0, loop_num) as i:
    # for_body

with bp.if_scope(condition):
    # if stmt
with bp.else_scope():
    # else stmt
```

创建了循环次数为 loop_num 的 for 循环和 if/else 判断语句。

使用示例

下面的代码展示了如何使用张量搬运、张量计算、循环控制接口，描述张量加的分块逻辑:

```
with bp.for_range(0, loop_num) as i:
    start = task_id * core_wl + i * loop_wl
    stop = start + loop_wl
    bp.memcpy(tensor_in0_n, tensor_in0[start:stop])
    bp.memcpy(tensor_in1_n, tensor_in1[start:stop])
    bp.add(tensor_out_n, tensor_in0_n, tensor_in1_n)
    bp.memcpy(tensor_out[start:stop], tensor_out_n)
```


上面的代码中，每个核处理的数据量为 core_wl, 核内每次循环处理的数据量为 loop_wl, 循环次数为 loop_num，task_id 索引核，i 索引循环次数。

6.1.2.3 编译

调用编译接口生成可执行 module fvec_add。如下所示：

```
fvec_add = bp.BuildBANG(inputs=[tensor_in0, tensor_in1], outputs=[tensor_out], kernel_name=
↪"fvec_add")
```

用户可以通过如下方式，保存 CPU 端 module 到目录 dirname 中的 host.so 文件，保存 MLU 端 module 到目录 dirname 中的 device.mlu 文件:

```
fvec_add.save(dirname)
```

用户可以通过如下方式载入运行：

```
from bangpy import load_module
fadd = load_module(dirname, target)

# 运行验证
fadd(a, b, c)
```

注意:

每个算子会保存独立的算子信息，因此用户需要为每个 BuildBANG 得到的 module 指定一个独立的空目录。

编译生成的可执行 module 会占用算子文件和目录的资源，因此用户在删除算子文件和目录之前，需要确保可执行 module 已经释放。

6.1.2.4 运行验证

这里用随机产生的输入数据和 numpy 的计算结果作为参考，验证结果正确性。如下所示：

```
in_a = bangpy.Array(np.random.uniform(size=shape).astype(dtype), target)
in_b = bangpy.Array(np.random.uniform(size=shape).astype(dtype), target)
out_c = bangpy.Array(np.zeros(shape, dtype=dtype), target)
fvec_add(in_a, in_b, out_c)
banpy.assert_allclose(out_c.asnumpy(), cpu_output, rtol=1e-2, atol=1e-2)
```


6.1.2.5 调试分析

采用如下两种方式进行调试分析；

- 插入打印语句，打印张量中某个元素的值，进行调试分析。

```
bp.print(tensor[index])
```

> **注意：**
> bp.print 接口仅支持 int32，float16 和 float32 数值类型。

- 在编译时加上 dump_ir 的选项，用来保存中间表示文件，进行调试分析。

```
f = bp.BuildBANG(inputs=[tensor_in0, tensor_in1], outputs=[tensor_out], dump_ir=True, kernel_
↪name="fvec_add")
```

6.2 TensorOp 算子开发流程

TensorOp 算子开发的主要步骤如下：

1. 计算描述：使用 TensorOp 接口描述计算，主要包括声明输入输出，调用计算接口等。
2. 编译：调用编译接口生成可执行 module，该可执行 module 可以保存成 CPU 端的 host.so 文件和 MLU 端的 device.mlu 文件。
3. 运行验证：用 numpy 生成输入数据，将数据传入可执行 module 中运行，检查算子的正确性。
4. 调试分析：针对运行结果，进行调试分析。

6.2.1 样例

实现反双曲正切的部分代码如下，完整代码参见安装目录/samples/tensor_op/test_atanh.py：

```
...
def run_cpu(data):
    return 0.5 * np.log((1 + data) / (1 - data))


def verify(*shape, dtype=bangpy.float16):
    # 声明 tensor
    input0 = tsop.tensor(shape, dtype, name='input0')
    # 描述计算
    out_atanh = 0.5 * tsop.log((1 + input0) / (1 - input0))
    # 生成可执行 module
    fmlu = tsop.BuildBANG([input0], [out_atanh], target=target)
```



```
    # 保存生成的算子文件到指定目录
    fmlu.save(path_to_save_ops)
    # 生成随机数并在 mlu 和 cpu 上测试
    data0 = generate_data(shape, dtype)
    ctx = bangpy.context(0)
    data_dev = bangpy.Array(data0, ctx)
    fmlu.set_input("input0", data_dev)
    fmlu.run()
    out_dev = fmlu.get_output(0)
    cpu_output = run_cpu(data0)
    # 比较 mlu 和 cpu 上运行结果
    bangpy.assert_allclose(out_dev.asnumpy(), cpu_output, rtol=1e-2, atol=1e-2)
    # 从保存的目录中载入算子文件运行验证
    floaded = bangpy.load_module(path_to_save_ops, target)
    floaded.set_input("input0", data0)
    floaded.run()
    mlu_output = floaded.get_output(0).asnumpy()
    bangpy.assert_allclose(mlu_output, cpu_output, rtol=1e-2, atol=1e-2)
```

6.2.2 代码解析

6.2.2.1 数据生成

用 numpy 生成输入数据。

```
np.random.uniform(size=shape, low=-0.9, high=0.9).astype(dtype)
```

6.2.2.2 计算描述

使用 TensorOp 接口描述计算，包括以下流程。

声明输入

声明输入需要给定输入的数值类型、形状、名字。例如：

```
from bangpy import tensor_op as tsop
input0 = tsop.tensor(shape, dtype, name='input0')
```

调用计算接口

调用 TensorOp 中的计算接口。例如：

```
out_atanh = 0.5 * tsop.log((1 + input0) / (1 - input0))
```

6.2.2.3 编译

调用编译接口可生成可执行 module fmlu。如下所示：

```
fmlu = tsop.BuildBANG([input0], [out_atanh], target=target)
```

用户可以通过如下方式，保存 CPU 端 module 到目录 dirname 中的 host.so 文件，保存 MLU 端 module 到目录 dirname 中的 device.mlu 文件:

```
fmlu.save(dirname)
```

用户可以通过如下方式载入运行：

```
from bangpy import load_module
fmlu = load_module(dirname, target=target)

# 运行验证
fmlu.set_input(input0=data0)
fmlu.run()
mlu_output = fmlu.get_output(0).asnumpy()
bangpy.assert_allclose(mlu_output, cpu_output, rtol=1e-2, atol=1e-2)
```

注意:

每个算子会保存独立的算子信息，因此用户需要为每个 BuildBANG 得到的 module 指定一个独立的空目录。

编译生成的可执行 module 会占用算子文件和目录的资源，因此用户在删除算子文件和目录之前，需要确保可执行 module 已经释放。

6.2.2.4 运行验证

这里用随机产生的输入数据和 numpy 的计算结果作为参考，验证结果正确性。如下所示：

```
data_dev = bangpy.Array(data0, ctx)
fmlu.set_input(input0=data_dev)
out_dev = fmlu.get_output(0)
cpu_output = run_cpu(data0)
bangpy.assert_allclose(out_dev.asnumpy(), cpu_output, rtol=1e-2, atol=1e-2)
```

6.2.2.5 调试分析

在编译时加上 dump_ir 的选项，用来保存中间表示文件，进行调试分析。

```
fmlu = tsop.BuildBANG([input0], [out_atanh], dump_ir=True, target=target)
```

7 TCP 接口

7.1 张量接口

7.1.1 创建张量

Tensor(

shape,

dtype,

name,

scope

)

在 TCP 容器中创建一个张量。

参数说明

- shape: 张量形状。类型: tuple of int。
- dtype: 张量元素的数值类型，参见数值类型 节。
- name: 张量名字，类型: str。
- scope: 存储空间类型（storage scope），取值范围包括 `nram`，`wram`，`global`，类型:str。

返回值描述

- 创建的张量，类型: Tensor。

注意事项

- scope 的取值范围包括 `nram`，`wram`，`global`。

示例

```
import bangpy
from bangpy import tcp
bp = tcp.TCP()  # 创建一个 TCP 容器，在该容器中进行张量操作，后面的示例代码中省略此创建过程。
tensor = bp.Tensor(shape=(n, h, w, c), name='input_tensor', dtype=bangpy.float16, scope="global
↪")
```

7.1.2 张量的属性

7.1.2.1 dtype

dtype

获取张量元素的数值类型。

参数说明

- 无。

返回值描述

- 张量元素的数值类型，参见数值类型 节。

注意事项

- dtype 是张量的只读属性。

示例

```
tensor = bp.Tensor(shape=(n, h, w, c), name='input_tensor', dtype=bangpy.float16, scope="global
↪")
assert tensor.dtype == bangpy.float16
```

7.1.2.2 name

name

获取张量的名字。

参数说明

- 无。

返回值描述

- 张量的名字。类型: str。

注意事项

- name 是张量的只读属性。

示例

```
tensor = bp.Tensor(shape=(n, h, w, c), name='input_tensor', dtype=bangpy.float16, scope="global
↪")
print(tensor.name) # output: input_tensor
```

7.1.2.3 scope

scope

获取张量的存储空间类型。

参数说明

- 无。

返回值描述

- 张量的存储空间类型。类型: str。

注意事项

- scope 的取值范围包括 `nram`，`wram`，`global`。
- scope 是张量的只读属性。

示例

```
tensor = bp.Tensor(shape=(n, h, w, c), name='input_tensor', dtype=bangpy.float16, scope="global
↪")
print(tensor.scope) # output: global
```

7.1.2.4 ndim

ndim

获取张量的维度数。

参数说明

- 无。

返回值描述

- 张量的维度数。类型: int。

注意事项

- ndim 是张量的只读属性。

示例

```
tensor = bp.Tensor(shape=(n, h, w, c), name='input_tensor', dtype=bangpy.float16, scope="global
↪")
print(tensor.ndim) # output: 4
```

7.1.2.5 shape

shape

获取张量的形状。

参数说明

- 无。

返回值描述

- 张量的形状。类型: tuple of int。

注意事项

- shape 是张量的只读属性。

示例

```
tensor = bp.Tensor(shape=(1, 56, 56, 256), name='input_tensor', dtype=bangpy.float16, scope=
↪"global")
print(tensor.shape) # output: (1, 56, 56, 256)
```

7.1.2.6 size

size

获取张量的大小。

参数说明

- 无。

返回值描述

- 张量的大小。类型: int。

注意事项

- size 是张量的只读属性。

示例

```
tensor = bp.Tensor(shape=(1, 56, 56, 256), name='input_tensor', dtype=bangpy.float16, scope=
↪"global")
print(tensor.size) # output: 802816
```

7.1.2.7 is_contiguous

is_contiguous

判断张量是否连续存储。

参数说明

- 无。

返回值描述

- 张量是否连续存储。类型: bool。

注意事项

- is_contiguous 是张量的只读属性。

示例

```
tensor = bp.Tensor(shape=(1, 56, 56, 256), name='input_tensor', dtype=bangpy.float16, scope=
↪"global")
print(tensor.is_contiguous) # output: True
print(tensor[:, :, :, :128].is_contiguous) # output: False
```

7.1.3 张量的方法

7.1.3.1 reshape

reshape(

new_shape

)

改变张量的形状。

参数说明

- new_shape: 新的形状。类型: tuple of int, tuple of Scalar。

返回值描述

- 拥有指定形状的张量。类型: Tensor。

注意事项

- reshape 只能改变连续存储的张量的形状。
- reshape 不会改变张量的数据排布。
- 返回值所描述的张量和原张量共享数据元素，如果需要拷贝数据元素请使用 memcpy 接口。

示例

```
tensor = bp.Tensor(shape=(1, 56, 56, 256), name='input_tensor', dtype=bangpy.float16, scope=
↪"global")
new_tensor = tensor.reshape((256, 3136))
```

7.1.3.2 flatten

flatten()

将张量的形状展平。

参数说明

- 无。

返回值描述

- 展平的一维张量。类型: Tensor。

注意事项

- flatten 只能展平连续存储的张量的形状。
- flatten 不会改变张量的数据排布。
- 返回值所描述的张量和原张量共享数据元素，如果需要拷贝数据元素请使用 memcpy 接口。

示例

```
tensor = bp.Tensor(shape=(1, 56, 56, 256), name='input_tensor', dtype=bangpy.float16, scope=
↪"global")
new_tensor = tensor.flatten()
```

7.1.3.3 reinterpret_cast

reinterpret_cast(

dtype

)

用新的数值类型解释张量。

参数说明

- dtype: 张量元素的数值类型，参见数值类型 节。

返回值描述

- 重新解释的张量。类型: Tensor。

注意事项

- 返回值所描述的张量和原张量共享数据元素，如果需要拷贝数据元素请使用 memcpy 接口。
- 返回值所描述的张量和原张量数值类型不同，但是存储比特位相同。

示例

```
tensor = bp.Tensor(shape=(1, 56, 56, 256), name='input_tensor', dtype=bangpy.float16, scope=
↪"global")
new_tensor = tensor.reinterpret_cast(bangpy.int8)
```

7.1.3.4 __getitem__

__getitem__(

index

)

索引并读取张量中的元素。

参数说明

- index: 索引值。类型: int, Scalar, Tensor, slice。

返回值描述

- 索引得到的元素组成的张量。类型: Tensor。

注意事项

- 当索引值是 slice 类型时，索引 slice 的 step 必须是 1。
- 当索引值是 Tensor 类型时，索引张量的大小必须是 1。

示例

```
tensor = bp.Tensor(shape=(1, 56, 56, 256), name='input_tensor', dtype=bangpy.float16, scope=
↪"global")
print(tensor[0][0])
# output: Tensor("tensor", shape=(56, 256), dtype=info(dtype=float16), scope=global,␣
↪buffer=buffer(tensor, 0x1ad8000))
print(tensor[0][0][0][0])
# output: Tensor("tensor", shape=(), dtype=info(dtype=float16), scope=global,␣
↪buffer=buffer(tensor, 0x1ad8000))
print(tensor[0][0:1])
# output: Tensor("tensor", shape=(1, 56, 256), dtype=info(dtype=float16, scope=global,␣
↪buffer=buffer(tensor, 0x1ad8000))
print(tensor[0][0:28])
```

```
# output: Tensor("tensor", shape=(28, 56, 256), dtype=info(dtype=float16, scope=global,␣
↪buffer=buffer(tensor, 0x1ad8000))
print(tensor[:, 1:2, 2:3, 4:5])
# output: Tensor("tensor", shape=(1, 1, 1, 1), dtype=info(dtype=float16, scope=global,␣
↪buffer=buffer(tensor, 0x1ad8000))
# print(tensor[0][0:9:2]) # 不支持 step 不是 1 的索引
# print(tensor[0][::2]) # 不支持 step 不是 1 的索引
```

7.1.3.5 __setitem__

__setitem__(

index,

value

)

索引并赋值或改变张量中的元素。

参数说明

- index: 索引值。类型: int, Scalar, Tensor, slice。
- value: 赋值的右值。类型: int, float, Scalar, bang.Tensor。

返回值描述

- 无。

注意事项

- 当索引值是 slice 类型时，索引 slice 的 step 必须是 1。
- 当索引值和赋值的右值是 Tensor 类型时，索引张量的大小必须是 1。

示例

```
tensor = bp.Tensor(shape=(1, 56, 56, 256), name='input_tensor', dtype=bangpy.float16, scope=
↪"global")
tensor[0, 0, 0, 0] = 1
tensor[0][0][0][0] = tensor[0, 0, 0, 1]
x = bp.Scalar(name='x', dtype=bangpy.float16, value=1)
tensor[0, 0, 0, 0] = x
```

7.2 标量接口

7.2.1 创建标量

Scalar(

dtype,

name,

value=None

)

在 TCP 容器中创建一个标量。

参数说明

- dtype: 标量的数值类型，参见数值类型 节。
- name: 标量名字。类型: str。
- value: 标量的初始值，即标量在定义时存储的值。当初始值为空时，表示声明一个标量。类型: int、float、Scalar、Tensor。

返回值描述

- 创建的标量。类型: Scalar。

注意事项

- 当 value 类型为 Tensor 时，其长度必须为 1。

示例

```
scalar = bp.Scalar(name='input_scalar', dtype=bangpy.float16, value=6)
```

7.2.2 标量的属性

7.2.2.1 dtype

获取标量的元素数值类型（参见数值类型 节）。

示例

```
scalar = bp.Scalar(name='input_scalar', dtype=bangpy.float16, value=6)
assert scalar.dtype == bangpy.float16
```

7.2.2.2 name

获取标量的名字，类型: str。

示例

```
scalar = bp.Scalar(name='input_scalar', dtype=bangpy.float16, value=6)
print(scalar.name) # output: input_scalar
```

7.2.3 标量的方法

7.2.3.1 assign

assign(

value

)

通过 assign 接口来设置或者改变标量的值。

参数说明

- value: 赋值的右值。类型: int、float、Scalar、Tensor。

返回值描述

- 无返回值。

注意事项

- 当 value 类型为 Tensor 时，其长度必须为 1。

示例

```
scalar = bp.Scalar(name='input_scalar', dtype=bangpy.float16, value=None)
scalar.assign(6)
```

7.2.3.2 astype

astype(

dtype,

rounding=None

)

对原标量进行数值类型转换，产生一个具有新的数值类型的副本标量。

参数说明

- dtype: 新的副本标量的数值类型。类型：int8, int16, int32, uint16, uint32, float16, float32。
- rounding: 舍入类型，包含有: rd: 四舍五入; dn: 向下取整; up: 向上取整; oz: 远离零取整; tz: 向零取整，类型：str。参见张量数值类型转换算子 。

返回值描述

- 新的副本张量。类型: Scalar。

注意事项

- astype 方法不会改变原标量的信息，而是返回一个新的标量。
- 只有 float16，float32 类型的标量转换为 int16，int32 类型时可以设置舍入参数 rounding。

示例

```
scalar = bp.Scalar(name='input_scalar', dtype=bangpy.float16, value=1.1)
scalar.astype(bangpy.int16, rounding="rd")
```

7.2.3.3 cast

cast(

dtype

)

用新的数值类型来解释标量，产生一个具有新的数值类型的副本标量。

参数说明

- dtype: 新的副本标量的数值类型。类型：int8, int16, int32, uint16, uint32, float16, float32。

返回值描述

- 新的副本张量。类型: Scalar。

注意事项

- cast 方法不会改变原标量的信息，而是返回一个新的标量。
- 新标量的值是用新的数值类型解释的原标量的值 (值的字节存储不变)，两者数值不同。

示例

```
scalar = bp.Scalar(name='input_scalar', dtype=bangpy.float16, value=1.1)
scalar.cast(bangpy.int16)
```

7.3 张量计算接口

注意:

除了数据搬运算子，本节的其他所有算子接口的操作数张量都必须连续存储。

7.3.1 数据搬运算子

7.3.1.1 memcpy

memcpy(

dst_data,

src_data,

dst_cluster=None

)

实现数据搬运，包括将 GDRAM 中的数据搬运到 NRAM 中，将 NRAM 中的数据搬运到 GDRAM 中，及将 NRAM 中的数据缓存到 NRAM 中。

参数说明

- dst_data: 目的操作数，类型: Tensor。
- src_data: 源操作数，类型: Tensor。
- dst_cluster: 目的 cluster，默认为 None，当在 SRAM 之间拷贝数据时 dst_cluster 有值。类型: int。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: int8, int16, int32, uint16, uint32, float16, float32。
- dst_data 和 src_data 的形状必须相同。

示例

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype=bangpy.float32, scope="global")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")


bp.memcpy(Output, Input)
```

7.3.2 内存初始化算子

7.3.2.1 assign

assign(

dst_data,

value

)

初始化 GDRAM 和 NRAM 中的内存。

参数说明

- dst_data: 目的操作数，要被初始化的张量。
- value: 用于初始化的数，可以是标量或立即数。

返回值描述

- 无返回值。

注意事项

- value 中支持的数值类型有: float16, float32, int16, int32。

示例

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype=bangpy.float32, scope="global")
bp.assign(Input, 1)
```

7.3.3 双目运算算子

7.3.3.1 add

add(

output,

input1,

input2,

cycle=False

)

对输入的数据进行加法运算，包括循环加法运算。可以是两个张量相加，也可以是张量和标量相加。

两个张量相加: $input1 = (m1, m2, m3), input2 = (n1, n2, n3)$

计算得到: $output = (m1 + n1, m2 + n2, m3 + n3)$

张量和标量相加: $input1 = (m1, m2, m3), input2 = n1$

计算得到: $output = (m1 + n1, m2 + n1, m3 + n1)$

循环加法运算: $input1 = (m1, m2, m3, m4), input2 = (n1, n2)$

计算得到: $output = (m1 + n1, m2 + n2, m3 + n1, m4 + n2)$

参数说明

- output: 输出张量，存储两个输入张量逐元素相加后的结果，数值类型和源操作数相同。
- input1, input2: 源操作数，可以是两个都是张量，也可以是一个张量，一个标量。当循环相加时(cycle=True)，input1 的数据长度必须能整除 input2 的数据长度。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, float32。
- 输入输出的数据长度必须 128 字节对齐，且输入输出张量的形状和数值类型一致。

示例

- 两个张量相加：

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(a,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.add(Output, Input1, Input2)
```

- 张量和标量相加：

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = be.Scalar(name='Input2', dtype=bangpy.float32, value=c)
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.add(Output, Input1, Input2)
```

- 循环相加：

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(1,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")
```



```
bp.add(Output, Input1, Input2, cycle=True)
```

7.3.3.2 divide

divide(

output,

input1,

input2

)

对输入张量进行除法运算。

参数说明

- output: 输出张量，存储两个输入张量逐元素相除得到的结果，数值类型和源操作数相同。
- input1, input2: 源操作数，两个相除的张量。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, float32。
- 输入输出数据都要 128 字节对齐，且输入输出张量的形状和数值类型一致。
- 除数的数据大小范围为：(0.03, 500]。

示例

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(a,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.divide(Output, Input1, Input2)
```


7.3.3.3 multiply

multiply(

output,

input1,

input2,

cycle=False

)

对输入张量进行乘法运算。支持循环乘法运算。

参数说明

- output: 输出张量，存储数据相乘后的结果，数值类型和源操作数相同。
- input1, input2: 源操作数，两个相乘的张量，或者一个张量和一个标量相乘。当循环相乘时(cycle=True)，input1 的数据长度必须能整除 input2 的数据长度。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, float32。
- 输入输出数据都要 128 字节对齐。

示例

- 两个张量相乘：

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(a,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.multiply(Output, Input1, Input2)
```

- 张量和标量相乘：

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Scalar(name='Input2', dtype=bangpy.float32, value=c)
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.multiply(Output, Input1, Input2)
```

- 循环相乘

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(1,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.multiply(Output, Input1, Input2, cycle=True)
```

7.3.3.4 subtract

subtract(

output,

input1,

input2,

cycle=False

)

对输入张量进行减法运算。支持循环减法运算

参数说明

- output: 输出张量，存储输入数据相减得到的结果，数值类型和源操作数相同。
- input1: 源操作数，被减数张量。
- input2: 源操作数，减数，可以是张量或标量。当循环相减时 (cycle=True)，input1 的数据长度必须能整除 input2 的数据长度。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, float32。
- 输入输出数据都要 128 字节对齐。

示例

- 两个张量相减：

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(a,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.subtract(Output, Input1, Input2)
```

- 张量和标量相减：

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = be.Scalar(name='Input2', dtype=bangpy.float32, value=c)
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.subtract(Output, Input1, Input2)
```

- 循环相减：

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(1,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.subtract(Output, Input1, Input2, cycle=True)
```

7.3.3.5 take

take(

dst_data,

src_data,

mask

)

根据 mask 里的数据对 src_data 中的数据进行逐个挑选，当 mask 里的数据不为零时，选取出对应的 src_data 中的数据。

参数说明

- dst_data: 挑选之后的数据，类型: Tensor。
- src_data: 源操作数，类型: Tensor。
- mask: 用于挑选的模板，类型: Tensor。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型有: float16, float32。
- 输入输出的张量元素个数必须是 64 的倍数。
- 输入、输出和 mask 张量必须定义在 NRAM 空间上。

示例

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype=bangpy.float32, scope="nram")
mask = bp.Tensor(shape=(a,b), name='mask', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.take(Output, Input, mask)
```

7.3.3.6 take_bitindex

take_bitindex(

dst_data,

src_data,

bitmask

)

根据 bitmask 里的数据对 src_data 中的数据进行逐个挑选，当 bitmask 里的数据不为零时，选取出对应的 src_data 中的数据。

参数说明

- dst_data: 挑选之后的数据，类型: Tensor。
- src_data: 源操作数，类型: Tensor。
- bitmask: 用于挑选的模板，类型: Tensor。

返回值描述

- 无返回值。

注意事项

- 输入输出支持的数值类型有: float16, float32，bitmask 支持所有类型。
- bitmask 中存的是 bit 值，根据 bit 值来挑选输入张量对应的值。
- 输入输出张量的元素个数必须是 1024 的倍数。
- 输入、输出和 bitmask 张量必须定义在 NRAM 空间上。

示例

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype=bangpy.float32, scope="nram")
bitmask = bp.Tensor(shape=(a,b), name='bitmask', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.take_bitindex(Output, Input, bitmask)
```


7.3.4 单目运算算子

7.3.4.1 abs

abs(

dst_data,

src_data

)

对输入张量中每个元素取绝对值。

参数说明

- dst_data: 对输入张量中每个元素取绝对值后的结果，数值类型和源操作数相同，类型: Tensor。
- src_data: 源操作数，类型: Tensor。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, float32。
- 输入输出张量必须 128 字节对齐且形状相同。

示例

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.abs(Output, Input)
```

7.3.4.2 cos

cos(

dst_data,

src_data,

aux1=None,

aux2=None,

mode=”taylor3”

)

cosine 余弦函数。

参数说明

- dst_data: 目的操作数，类型: Tensor。
- src_data: 源操作数，类型: Tensor。
- aux1,aux2: 计算用的辅助空间，默认为 None，类型: Tensor。
- mode: 多阶泰勒展开运算模式，默认为泰勒三阶展开: “taylor3”，可选的模式有泰勒三阶展开: “taylor3”与泰勒四阶展开: “taylor4”，类型: str。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, float32。
- 输入输出数据必须 128 字节对齐且长度，数值类型相同。
- aux1,aux2 必须和 src_data 形状一致，数值类型一致。
- 使用多阶泰勒展开运算模式时，计算辅助空间 aux1 与 aux2 均不能为空，src_data 的数据大小范围为: [-7.75,7.75]。其他模式下 src_data 的数据大小范围为: [-6.28,6.28]。

示例

- 普通余弦运算:

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.cos(Output, Input)
```

- 多阶泰勒展开余弦运算:

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype=bangpy.float32, scope="nram")
Aux1 = bp.Tensor(shape=(a,b), name='Aux1', dtype=bangpy.float32, scope="nram")
Aux2 = bp.Tensor(shape=(a,b), name='Aux2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.cos(Output, Input, Aux1, Aux2)
```


7.3.4.3 exp

exp(

dst_data,

src_data,

mode=””

)

对输入张量中每个元素进行幂运算，底数为 e。

例如: $src\_data = (m1, m2, m3)$

计算得到: $dst\_data = (e^{m1}, e^{m2}, e^{m3})$

参数说明

- dst_data: 对输入张量中每个元素进行幂运算之后的结果，数值类型和源操作数相同，类型: Tensor。
- src_data: 源操作数，类型: Tensor。
- mode: 运算模式，默认为””，还可以是 high_precision 或 less_0，类型: str。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, float32。高精度模式 (mode=high_precision) 时，float16 误差比较大，推荐使用 float32。
- src_data 的数据大小范围为: [-7.75,10]。
- 输入输出数据必须 128 字节对齐且形状相同。

示例

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")


bp.exp(Output, Input)
```

7.3.4.4 exp2

exp2(

dst_data,

src_data,

)

对输入张量中每个元素进行幂运算，底数为 2。

例如: $src\_data = (m1, m2, m3)$

计算得到: $dst\_data = (2^{m1}, 2^{m2}, 2^{m3})$

参数说明

- dst_data: 对输入张量中每个元素进行幂运算之后的结果，数值类型和源操作数相同，类型: Tensor。
- src_data: 源操作数，类型: Tensor。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, float32。
- src_data 的数据大小范围为: [0, 31]。
- 输入输出数据必须 128 字节对齐且形状相同。

示例

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.exp2(Output, Input)
```

7.3.4.5 gelu

gelu(

dst_data,

src_data,

high_precision=False

)

对输入张量中每个元素进行高斯误差线性单元运算 (GELU)。

参数说明

- dst_data: 对输入张量中每个元素进行运算之后的结果，数值类型和源操作数相同，类型: Tensor。
- src_data: 源操作数，类型: Tensor。
- high_precision: 是否开启高精度模式，类型: bool，取值为 True 或 False。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, float32。
- 输入输出数据必须 128 字节对齐且形状相同。
- 高精度模式 (mode=high_precision) 时，误差较大，推荐使用普通模式。

示例

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.gelu(Output, Input)
```

7.3.4.6 log

log(
dst_data,
src_data,
high_precision=False
)

对输入张量中每个元素进行底数为 e 的对数运算。

参数说明

- dst_data: 对输入张量中每个元素进行以 e 为底数的对数运算，数值类型和源操作数相同，类型: Tensor。
- src_data: 源操作数，类型: Tensor。
- high_precision: 是否开启高精度模式，类型: bool，取值为 True 或 False。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, float32，在 float16 下误差比较大，推荐使用 float32。

- src_data 的数据大小范围为: [0.00002,60000]。
- 输入输出数据必须 128 字节对齐且形状相同。

示例

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.log(Output, Input)
```

7.3.4.7 reciprocal

reciprocal(

dst_data,

src_data,

mode="")

对输入张量中每个元素求倒数。

例如: $src\_data = (m1, m2, m3)$

计算得到: $dst\_data = (1/m1, 1/m2, 1/m3)$

参数说明

- dst_data: 对输入张量中每个元素求倒后的结果，数值类型和源操作数相同，类型: Tensor。
- src_data: 源操作数，类型: Tensor。
- mode: 运算模式，默认为""，还可以是 high_precision 或 greater_1，类型: str。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, float32。
- 当运算模式为 greater_1 时，src_data 的数据大小范围为: [0.00002,60000]。其他模式下，src_data 的数据大小范围为: (0.01, 500]。
- 输入输出数据必须 128 字节对齐且形状相同。

示例

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")
```

```
bp.reciprocal(Output, Input)
```

7.3.4.8 relu

relu(

dst_data,

src_data

)

对输入张量中每个元素进行 relu 运算。

例如: $src\_data = (m1, m2, m3)$#$m1, m2$为正数，$m3$为负数

计算得到: $dst\_data = (m1, m2, 0)$

参数说明

- dst_data: 输出数据，如果输入元素为非负数，对应的输出元素等于输入元素，如果为负数，输出元素就为 0。数值类型和源操作数相同，类型: Tensor。
- src_data: 源操作数，类型: Tensor。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, float32。
- 输入输出数据必须 128 字节对齐且形状相同。

示例

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.relu(Output, Input)
```

7.3.4.9 rsqrt

rsqrt(

dst_data,

src_data,

high_precision=False

)

对输入张量中每个元素，先开方再求倒数。

例如: $src\_data = (m1, m2, m3)$

计算得到: $dst\_data = (1/\sqrt{m1}, 1/\sqrt{m2}, 1/\sqrt{m3})$

参数说明

- dst_data: 输出数据，数值类型和源操作数相同，类型: Tensor。
- src_data: 源操作数，类型: Tensor。
- high_precision: 是否开启高精度模式，类型: bool，取值为 True 或 False。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, float32。
- src_data 的数据大小范围为: [0.005,63487]。
- 输入输出数据必须 128 字节对齐且形状相同。

示例

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.rsqrt(Output, Input)
```

7.3.4.10 sigmoid

sigmoid(

dst_data,

src_data,

aux1=None,

aux2=None,

mode=”taylor3”

)

对输入张量中每个元素进行 sigmoid 运算，可将输入数据映射到 (0,1) 的区间。

例如: $src\_data = (m1, m2, m3)$

计算得到: $dst\_data = (1/(1+e^{-m1}), 1/(1+e^{-m2}), 1/(1+e^{-m3}))$

参数说明

- dst_data: 目的操作数，类型: Tensor。
- src_data: 源操作数，类型: Tensor。
- aux1,aux2: 计算用的辅助空间，默认为 None，类型: Tensor。
- mode: 多阶泰勒展开运算模式，默认为泰勒三阶展开: “taylor3”，可选的模式有泰勒三阶展开: “taylor3”与泰勒四阶展开: “taylor4”，类型: str。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, float32。
- 输入输出数据必须 128 字节对齐且长度，数值类型相同。
- aux1,aux2 必须和 src_data 形状一致，数值类型一致。
- 使用多阶泰勒展开运算模式时，计算辅助空间 aux1 与 aux2 均不能为空，src_data 的数据大小范围为: [-7.75,7.75]。

示例

- 普通运算：

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.sigmoid(Output, Input)
```

- 多阶泰勒展开运算：

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype=bangpy.float32, scope="nram")
Aux1 = bp.Tensor(shape=(a,b), name='Aux1', dtype=bangpy.float32, scope="nram")
Aux2 = bp.Tensor(shape=(a,b), name='Aux2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.sigmoid(Output, Input, Aux1, Aux2)
```


7.3.4.11 sqrt

sqrt(

dst_data,

src_data,

high_precision=False

)

对输入张量中每个元素进行开方运算。

例如: $src\_data = (m1, m2, m3)$

计算得到: $dst\_data = (sqrtm1, sqrtm2, sqrtm3)$

参数说明

- dst_data: 输出数据，数值类型和源操作数相同，类型: Tensor。
- src_data: 源操作数，类型: Tensor。
- high_precision: 是否开启高精度模式，类型: bool，取值为 True 或 False。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, float32。
- src_data 的数据大小范围为: [0,65504]。
- 输入输出数据必须 128 字节对齐且形状相同。

示例

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.sqrt(Output, Input)
```

7.3.4.12 square

square(

dst_data,

src_data

)

对输入张量中每个元素求平方。

参数说明

- dst_data: 输出数据，数值类型和源操作数相同，类型: Tensor。
- src_data: 源操作数，类型: Tensor。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, float32。
- 输入输出张量的元素个数必须是 64 的倍数。

示例

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.square(Output, Input)
```

7.3.4.13 sign

sign(

dst_data,

src_data

)

对输入张量中每个元素进行符号 (sign) 函数表示处理。

参数说明

- dst_data: 输出数据，如果输入元素为非负数，对应的输出元素等于 1.0，如果为负数，输出元素就为-1.0。数值类型和源操作数相同，类型: Tensor。
- src_data: 源操作数，类型: Tensor。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, float32。
- 输入输出数据必须 128 字节对齐且形状相同。

示例

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.sign(Output, Input)
```

7.3.4.14 sin

sin(

dst_data,

src_data,

aux1=None,

aux2=None,

mode=”taylor3”

)

sine 正弦函数。

参数说明

- dst_data: 目的操作数，类型: Tensor。
- src_data: 源操作数，类型: Tensor。
- aux1,aux2: 计算用的辅助空间，默认为 None，类型: Tensor。
- mode: 多阶泰勒展开运算模式，默认为泰勒三阶展开: “taylor3”，可选的模式有泰勒三阶展开: “taylor3”与泰勒四阶展开: “taylor4”，类型: str。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, float32。
- 输入输出数据必须 128 字节对齐且长度，数值类型相同。
- aux1,aux2 必须和 src_data 形状一致，数值类型一致。
- 使用多阶泰勒展开运算模式时，计算辅助空间 aux1 与 aux2 均不能为空，src_data 的数据大小范围为: [-7.75,7.75]。其他模式下 src_data 的数据大小范围为: [-6.28,6.28]。

示例

- 普通正弦运算:

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")
```

```
bp.sin(Output, Input)
```

- 多阶泰勒展开正弦运算：

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype=bangpy.float32, scope="nram")
Aux1 = bp.Tensor(shape=(a,b), name='Aux1', dtype=bangpy.float32, scope="nram")
Aux2 = bp.Tensor(shape=(a,b), name='Aux2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.sin(Output, Input, Aux1, Aux2)
```

7.3.4.15 softplus

softplus(

dst_data,

src_data,

aux1,

aux2,

mode=”taylor3”

)

对输入张量进行逐元素 ReLU 运算后结果进行平滑近似处理 (softplus)。具体的数学公式为：

例如: $src\_data = (m1, m2, m3)$

计算得到: $dst\_data = (\log(1 + e^{m1}), \log(1 + e^{m2}), \log(1 + e^{m3}))$

参数说明

- dst_data: 目的操作数，类型: Tensor。
- src_data: 源操作数，类型: Tensor。
- aux1,aux2: 计算用的辅助空间，类型: Tensor。
- mode: 多阶泰勒展开运算模式，默认为泰勒三阶展开: “taylor3”，可选的模式有泰勒三阶展开: “taylor3”与泰勒四阶展开: “taylor4”，类型: str。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, float32。

- 输入输出数据必须 128 字节对齐且长度，数值类型相同。
- aux1,aux2 必须和 src_data 形状一致，数值类型一致。
- src_data 的数据大小范围为: [-7.75,7.75]。

示例

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype=bangpy.float32, scope="nram")
Aux1 = bp.Tensor(shape=(a,b), name='Aux1', dtype=bangpy.float32, scope="nram")
Aux2 = bp.Tensor(shape=(a,b), name='Aux2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.softplus(Output, Input, Aux1, Aux2)
```

7.3.4.16 tanh

tanh(

dst_data,

src_data,

aux1=None,

aux2=None,

mode=”taylor3”

)

对输入张量中每个元素进行 tanh 运算，可将输入映射到 (-1,1) 的区间:

例如: $src\_data = (m1, m2, m3)$

计算得到: $dst\_data = ((e^{m1} - e^{-m1})/(e^{m1} + e^{-m1}), (e^{m2} - e^{-m2})/(e^{m2} + e^{-m2}), (e^{m3} - e^{-m3})/(e^{m3} + e^{-m3}))$

参数说明

- dst_data: 目的操作数，类型: Tensor。
- src_data: 源操作数，类型: Tensor。
- aux1,aux2: 计算用的辅助空间，默认为 None，类型: Tensor。
- mode: 多阶泰勒展开运算模式，默认为泰勒三阶展开: “taylor3”，可选的模式有泰勒三阶展开: “taylor3”与泰勒四阶展开: “taylor4”，类型: str。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, float32。
- 输入输出数据必须 128 字节对齐且长度，数值类型相同。
- aux1,aux2 必须和 src_data 形状一致，数值类型一致。
- 使用多阶泰勒展开运算模式时，计算辅助空间 aux1 与 aux2 均不能为空，src_data 的数据大小范围为: [-7.75,7.75]。其他模式下 src_data 的数据大小范围为: (0.01, 500]。

示例

- 普通运算：

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.tanh(Output, Input)
```

- 多阶泰勒展开运算：

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype=bangpy.float32, scope="nram")
Aux1 = bp.Tensor(shape=(a,b), name='Aux1', dtype=bangpy.float32, scope="nram")
Aux2 = bp.Tensor(shape=(a,b), name='Aux2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.tanh(Output, Input, Aux1, Aux2)
```

7.3.4.17 rand

rand(

dst_data

)

对输入张量中每个元素进行随机数赋值，随机数范围为 [-32768, 32767]

参数说明

- dst_data: 输出张量，必须定义在 NRAM 空间上，类型: Tensor。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: int16。
- 输出张量元素个数必须被 64 整除。

示例

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype=bangpy.float32, scope="nram")

bp.rand(Output)
```

7.3.4.18 zeros

zeros(

dst_data

)

给张量赋初值 0。

参数说明

- dst_data: 输出张量，必须定义在 NRAM 空间上，类型: Tensor。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: int16。
- 输出张量元素个数必须被 64 整除。

示例

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype=bangpy.float32, scope="nram")

bp.zeros(Output)
```

7.3.5 比较运算算子

7.3.5.1 equal

equal(

output,

input1,

input2,

mode=””

)

对输入张量进行比较，判断是否: $input1 == input2$ 。mode 是支持设置的模式，共支持三种，mode=[“”，“cycle”，“bit_index”]。

参数说明

- output: 输出张量，存储数据比较后的结果，类型: Tensor。
- input1, input2: 源操作数，类型: Tensor 或 Scalar。

返回值描述

- 无返回值。

注意事项

- 输入和输出张量必须定义在 NRAM 空间上。
- 支持的数值类型: float16, float32。
- mode 模式说明:
  1) mode=“”，逐个元素比较:
     模式参数缺省值为空字符串，输入是两个张量，或者一个张量和一个标量。张量元素个数必须被 64 整除，输入张量与输出张量的 shape 一致，逐个元素进行比较，相等结果为 1，否则为 0，存入输出张量中。
  2) mode=“cycle”，循环比较:
     模式参数值为字符串“cycle”，输入是两个张量。两个张量的长度必须 128 字节对齐。第一个输入张量 input1 元素个数能够被第二个输入张量 input2 元素个数整除。
  3) mode=“bit_index”，逐个比特位比较:
     模式参数值为字符串“bit_index”，输入是两个张量。输出张量的比特位数与输入张量的元素个数相同，用每一个比特位表示对应位置张量元素的比较结果，相等则比特位位置为 1，否则为 0。输入张量的元素个数必须是 512 的倍数。

示例

- mode=“”:

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(a,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.equal(Output, Input1, Input2)

Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Scalar(name='Input2', dtype=bangpy.float32, value=c)
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.equal(Output, Input1, Input2)
```

- mode=“cycle”:

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(1,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.equal(Output, Input1, Input2, mode="cycle")
```

- mode=“bit_index”：

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(a,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.equal(Output, Input1, Input2, mode="bit_index")
```

7.3.5.2 greater

greater(

output,

input1,

input2,

mode=””

)

对输入张量进行比较，判断是否: $input1 > input2$ 。mode 是支持设置的模式，共支持三种，mode=[“” , “cycle” , “bit_index”]。

参数说明

- output: 输出张量，存储数据比较后的结果，类型: Tensor。
- input1, input2: 源操作数，类型: Tensor 或 Scalar。

返回值描述

- 无返回值。

注意事项

- 输入和输出张量必须定义在 NRAM 空间上。
- 支持的数值类型: float16, float32。
- mode 模式说明:

  1) mode=“”，逐个元素比较：

模式参数缺省值为空字符串，输入是两个张量，或者一个张量和一个标量。张量元素个数必须被 64 整除，输入张量与输出张量的 shape 一致，逐个元素进行比较，大于结果为 1，否则为 0，存入输出张量中。

2) mode=“cycle”，循环比较：

模式参数值为字符串“cycle”，输入是两个张量。两个张量的长度必须 128 字节对齐。第一个输入张量 input1 元素个数能够被第二个输入张量 input2 元素个数整除。

3) mode=“bit_index”，逐个比特位比较：

模式参数值为字符串“bit_index”，输入是两个张量。输出张量的比特位数与输入张量的元素个数相同，用每一个比特位表示对应位置张量元素的比较结果，大于则比特位位置为 1，否则为 0。输入张量的元素个数必须是 512 的倍数。

示例

- mode=“”：

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(a,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.greater(Output, Input1, Input2)

Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Scalar(name='Input2', dtype=bangpy.float32, value=c)
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.greater(Output, Input1, Input2)
```

- mode=“cycle”：

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(1,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.greater(Output, Input1, Input2, mode="cycle")
```

- mode=“bit_index”：

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(a,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.greater(Output, Input1, Input2, mode="bit_index")
```


7.3.5.3 greater_equal

greater_equal(

output,

input1,

input2,

mode=””

)

对输入张量进行比较，判断是否: $input1 >= input2$ 。mode 是支持设置的模式，共支持三种，mode=[“”，“cycle”，“bit_index”]。

参数说明

- output: 输出张量，存储数据比较后的结果，类型: Tensor。
- input1, input2: 源操作数，类型: Tensor 或 Scalar。

返回值描述

- 无返回值。

注意事项

- 输入和输出张量必须定义在 NRAM 空间上。
- 支持的数值类型: float16, float32。
- mode 模式说明:
  1) mode=“”，逐个元素比较：
     模式参数缺省值为空字符串，输入是两个张量，或者一个张量和一个标量。张量元素个数必须被 64 整除，输入张量与输出张量的 shape 一致，逐个元素进行比较，大于等于结果为 1，否则为 0，存入输出张量中。
  2) mode=“cycle”，循环比较：
     模式参数值为字符串“cycle”，输入是两个张量。两个张量的长度必须 128 字节对齐。第一个输入张量 input1 元素个数能够被第二个输入张量 input2 元素个数整除。
  3) mode=“bit_index”，逐个比特位比较
     模式参数值为字符串“bit_index”, 输入是两个张量。输出张量的比特位数与输入张量的元素个数相同，用每一个比特位表示对应位置张量元素的比较结果，大于等于则比特位位置为 1，否则为 0。输入张量的元素个数必须是 512 的倍数。

示例

- mode=“”：

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(a,b), name='Input2', dtype=bangpy.float32, scope="nram")
```

```
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.greater_equal(Output, Input1, Input2)

Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Scalar(name='Input2', dtype=bangpy.float32, value=c)
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.greater_equal(Output, Input1, Input2)
```

- mode= “cycle”:

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(1,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.greater_equal(Output, Input1, Input2, mode="cycle")
```

- mode= “bit_index”:

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(a,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.greater_equal(Output, Input1, Input2, mode="bit_index")
```

7.3.5.4 less

less(

output,

input1,

input2,

mode=””

)

对输入张量进行比较，判断是否: $input1 < input2$ 。mode 是支持设置的模式，共支持三种，mode=[“” , “cycle” , “bit_index”]。

参数说明

- output: 输出张量，存储数据比较后的结果，类型: Tensor。

- input1, input2: 源操作数，类型: Tensor 或 Scalar。

返回值描述

- 无返回值。

注意事项

- 输入和输出张量必须定义在 NRAM 空间上。
- 支持的数值类型: float16, float32。
- mode 模式说明:

  1) mode=“”，逐个元素比较:

     模式参数缺省值为空字符串，输入是两个张量，或者一个张量和一个标量。张量元素个数必须被 64 整除，输入张量与输出张量的 shape 一致，逐个元素进行比较，小于结果为 1，否则为 0，存入输出张量中。

  2) mode=“cycle”，循环比较:

     模式参数值为字符串“cycle”，输入是两个张量。两个张量的长度必须 128 字节对齐。第一个输入张量 input1 元素个数能够被第二个输入张量 input2 元素个数整除。

  3) mode=“bit_index”，逐个比特位比较:

     模式参数值为字符串“bit_index”，输入是两个张量。输出张量的比特位数与输入张量的元素个数相同，用每一个比特位表示对应位置张量元素的比较结果，小于则比特位位置为 1，否则为 0。输入张量的元素个数必须是 512 的倍数。

示例

- mode=“”：

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(a,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.less(Output, Input1, Input2)

Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Scalar(name='Input2', dtype=bangpy.float32, value=c)
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.less(Output, Input1, Input2)
```

- mode=“cycle”：

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(1,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")
```



```
bp.less(Output, Input1, Input2, mode="cycle")
```

- mode=“bit_index”：

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(a,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.less(Output, Input1, Input2, mode="bit_index")
```

7.3.5.5 less_equal

less_equal(

output,

input1,

input2,

mode=””

)

对输入张量进行比较，判断是否: $input1 <= input2$ 。mode 是支持设置的模式，共支持三种，mode=[“”，“cycle”，“bit_index”]。

参数说明

- output: 输出张量，存储数据比较后的结果，类型: Tensor。
- input1, input2: 源操作数，类型: Tensor 或 Scalar。

返回值描述

- 无返回值。

注意事项

- 输入和输出张量必须定义在 NRAM 空间上。
- 支持的数值类型: float16, float32。
- mode 模式说明:

1) mode=“”，逐个元素比较：

模式参数缺省值为空字符串，输入是两个张量，或者一个张量和一个标量。张量元素个数必须被 64 整除，输入张量与输出张量的 shape 一致，逐个元素进行比较，小于等于结果为 1，否则为 0，存入输出张量中。

2) mode=“cycle”，循环比较：

模式参数值为字符串“cycle”，输入是两个张量。两个张量的长度必须 128 字节对齐。第一个输入张量 input1 元素个数能够被第二个输入张量 input2 元素个数整除。

3) mode=“bit_index”，逐个比特位比较：

模式参数值为字符串“bit_index”，输入是两个张量。输出张量的比特位数与输入张量的元素个数相同，用每一个比特位表示对应位置张量元素的比较结果，小于等于则比特位位置为 1，否则为 0。输入张量的元素个数必须是 512 的倍数。

示例

- mode=“”：

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(a,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.less_equal(Output, Input1, Input2)

Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Scalar(name='Input2', dtype=bangpy.float32, value=c)
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.less_equal(Output, Input1, Input2)
```

- mode=“cycle”：

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(1,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.less_equal(Output, Input1, Input2, mode="cycle")
```

- mode=“bit_index”：

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(a,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.less_equal(Output, Input1, Input2, mode="bit_index")
```


7.3.5.6 not_equal

not_equal(

output,

input1,

input2,

mode=””

)

对输入张量进行比较，判断是否: $input1 != input2$ 。mode 是支持设置的模式，共支持三种，mode=[“”，“cycle”，“bit_index”]。

参数说明

- output: 输出张量，存储数据比较后的结果，类型: Tensor。
- input1, input2: 源操作数，类型: Tensor 或 Scalar。

返回值描述

- 无返回值。

注意事项

- 输入和输出张量必须定义在 NRAM 空间上。
- 支持的数值类型: float16, float32。
- mode 模式说明:
  1) mode=“”，逐个元素比较：
     模式参数缺省值为空字符串，输入是两个张量，或者一个张量和一个标量。张量元素个数必须被 64 整除，输入张量与输出张量的 shape 一致，逐个元素进行比较，不相等结果为 1，否则为 0，存入输出张量中。
  2) mode=“cycle”，循环比较：
     模式参数值为字符串“cycle”，输入是两个张量。两个张量的长度必须 128 字节对齐。第一个输入张量 input1 元素个数能够被第二个输入张量 input2 元素个数整除。
  3) mode=“bit_index”，逐个比特位比较：
     模式参数值为字符串“bit_index”, 输入是两个张量。输出张量的比特位数与输入张量的元素个数相同，用每一个比特位表示对应位置张量元素的比较结果，不相等则比特位位置为 1，否则为 0。输入张量的元素个数必须是 512 的倍数。

示例

- mode=“”：

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(a,b), name='Input2', dtype=bangpy.float32, scope="nram")
```

```
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.not_equal(Output, Input1, Input2)

Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Scalar(name='Input2', dtype=bangpy.float32, value=c)
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.not_equal(Output, Input1, Input2)
```

- mode= “cycle” :

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(1,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.not_equal(Output, Input1, Input2, mode="cycle")
```

- mode= “bit_index” :

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(a,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.not_equal(Output, Input1, Input2, mode="bit_index")
```

7.3.5.7 maximum

maximum(

output,

input1,

input2,

mode=””

)

对输入张量中的元素进行逐个比较，选取两个数中较大的那个数并将结果存储在输出张量中。mode 是支持设置的模式，共支持两种，mode=[“” , “cycle”]。

参数说明

- output: 输出张量，存储数据比较后的结果，类型: Tensor。

- input1, input2: 源操作数，类型: Tensor 或 Scalar。

返回值描述

- 无返回值。

注意事项

- 输入和输出张量必须定义在 NRAM 空间上。
- 支持的数值类型: float16, float32。
- mode 模式说明:

  1) mode=“”，逐个元素比较:

     模式参数缺省值为空字符串，输入是两个张量，或者一个张量和一个标量。张量元素个数必须被 64 整除，输入张量与输出张量的 shape 一致，逐个元素进行比较，选取较大的元素存入输出张量中。

  2) mode=“cycle”，循环比较:

     模式参数值为字符串“cycle”，输入是两个张量。两个张量的长度必须 128 字节对齐。第一个输入张量 input1 元素个数能够被第二个输入张量 input2 元素个数整除。

示例

- mode=“”:

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(a,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.maximum(Output, Input1, Input2)

Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Scalar(name='Input2', dtype=bangpy.float32, value=c)
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.maximum(Output, Input1, Input2)
```

- mode=“cycle”:

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(1,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.maximum(Output, Input1, Input2, mode="cycle")
```


7.3.5.8 minimum

minimum(

output,

input1,

input2,

mode=””

)

对输入张量中的元素进行逐个比较，选取两个数中较小的那个数并将结果存储在输出张量中。mode 是支持设置的模式，共支持两种，mode=[“”，“cycle”]。

参数说明

- output: 输出张量，存储数据比较后的结果，类型: Tensor。
- input1, input2: 源操作数，类型: Tensor 或 Scalar。

返回值描述

- 无返回值。

注意事项

- 输入和输出张量必须定义在 NRAM 空间上。
- 支持的数值类型: float16, float32。
- mode 模式说明:
  1) mode=“”，逐个元素比较：
     模式参数缺省值为空字符串，输入是两个张量，或者一个张量和一个标量。张量元素个数必须被 64 整除，输入张量与输出张量的 shape 一致，逐个元素进行比较，选取较小元素存入输出张量中。
  2) mode=“cycle”，循环比较：
     模式参数值为字符串“cycle”，输入是两个张量。两个张量的长度必须 128 字节对齐。第一个输入张量 input1 元素个数能够被第二个输入张量 input2 元素个数整除。

示例

- mode=“”:

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(a,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.minimum(Output, Input1, Input2)
```

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Scalar(name='Input2', dtype=bangpy.float32, value=c)
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.minimum(Output, Input1, Input2)
```

- mode= “cycle”:

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(1,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.maximum(Output, Input1, Input2, mode="cycle")
```

7.3.6 逻辑运算算子

7.3.6.1 logical_and

logical_and(

output,

input1,

input2,

cycle=False

)

对输入的两个张量进行逻辑与运算。支持循环运算。当循环运算时 (cycle=True)，input1 的数据长度必须能整除 input2 的数据长度。

参数说明

- output: 输出张量，存储源操作数进行逻辑与运算后的结果，数值类型和源操作数相同。
- input1, input2: 源操作数，两个进行逻辑与运算的张量，数值类型相同。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, float32。
- 在进行非循环运算时，输入输出张量的元素个数必须是 64 的倍数。在进行循环运算时，输入输出张量的数据长度必须 128 字节对齐。

- 进行非循环运算时，输入输出张量的长度必须一致。

示例

- 普通相与运算：

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(a,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.logical_and(Output, Input1, Input2)
```

- 循环相与运算：

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(1,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.logical_and(Output, Input1, Input2, cycle=True)
```

7.3.6.2 logical_or

logical_or(

output,

input1,

input2,

cycle=False

)

对输入的两个张量进行逻辑或运算。支持循环运算。当循环运算时 (cycle=True)，input1 的数据长度必须能整除 input2 的数据长度。

参数说明

- output: 输出张量，存储源操作数进行逻辑或运算后的结果，数值类型和源操作数相同。
- input1, input2: 源操作数，两个进行逻辑或运算的张量，数值类型相同。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, float32。

- 在进行非循环运算时，输入输出张量的元素个数必须是 64 的倍数。在进行循环运算时，输入输出张量的数据长度必须 128 字节对齐。
- 进行非循环运算时，输入输出张量的长度必须一致。

示例

- 普通相或运算：

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(a,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.logical_or(Output, Input1, Input2)
```

- 循环相或运算：

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(1,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.logical_or(Output, Input1, Input2, cycle=True)
```

7.3.6.3 logical_xor

logical_xor(
output,
input1,
input2,
cycle=False
)

对输入的两个张量进行逻辑异或运算。支持循环运算。当循环运算时 (cycle=True)，input1 的数据长度必须能整除 input2 的数据长度。

参数说明

- output: 输出张量，存储源操作数进行逻辑异或运算后的结果，数值类型和源操作数相同。
- input1, input2: 源操作数，两个进行逻辑异或运算的张量，数值类型相同。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, float32。
- 在进行非循环运算时，输入输出张量的元素个数必须是 64 的倍数。在进行循环运算时，输入输出张量的数据长度必须 128 字节对齐。
- 进行非循环运算时，输入输出张量的长度必须一致。

示例

- 普通异或运算：

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(a,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.logical_xor(Output, Input1, Input2)
```

- 循环异或运算：

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="nram")
Input2 = bp.Tensor(shape=(1,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.logical_xor(Output, Input1, Input2, cycle=True)
```

7.3.6.4 logical_not

logical_not(
dst_data,
src_data
)

对输入张量中每个元素进行逻辑非运算。

参数说明

- dst_data: 对输入张量中每个元素进行逻辑非运算后的结果，数值类型和源操作数相同，类型: Tensor。
- src_data: 源操作数，类型: Tensor。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, float32。
- 输入输出张量长度必须 128 字节对齐且形状一致。

示例

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.logical_not(Output, Input)
```

7.3.7 归约类算子

7.3.7.1 sum

sum(

dst_data,

src_data

)

对源操作数每 128 字节的元素求和。

参数说明

- dst_data: 输出张量，每 128 字节内，第一个元素是对应的源操作数 128 字节求和后的结果，后面元素是 0。
- src_data: 源操作数，类型: Tensor。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型有: float16, float32。
- 输入输出的张量元素长度必须 128 字节对齐。当输入数值类型是 float16 时，对每 64 个数据求和，当输入数值类型是 float32 时，对每 32 个数据求和。

示例

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.sum(Output, Input)
```

7.3.7.2 first_nonzero

first_nonzero(

dst_data,

src_data

)

找到第一个源操作数的第一个非 0 数，并将 index 存在输出张量的第一个元素中，如果源操作数均为 0，则输出张量第一个元素中的值为-1。

参数说明

- dst_data: 输出张量，类型: Tensor。
- src_data: 源操作数，类型: Tensor。

返回值描述

- 无返回值。

注意事项

- 输入和输出张量必须定义在 NRAM 空间上。
- 支持的数值类型有: float16, float32。
- 输入输出张量的元素个数必须被 64 整除。

示例

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.first_nonzero(Output, Input)
```

7.3.7.3 last_nonzero

last_nonzero(

dst_data,

src_data

)

找到第一个源操作数的最后一个非 0 数，并将 index 存在输出张量的第一个元素中，如果源操作数均为 0，则输出张量第一个元素中的值为-1。

参数说明

- dst_data: 输出张量，类型: Tensor。

- src_data: 源操作数，类型: Tensor。

返回值描述

- 无返回值。

注意事项

- 输入和输出张量必须定义在 NRAM 空间上。
- 支持的数值类型有: float16, float32。
- 输入输出张量的元素个数必须被 64 整除。

示例

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.last_nonzero(Output, Input)
```

7.3.7.4 amax

amax(

dst_data,

src_data

)

选取输入张量中所有元素的最大值，并记录最大值的坐标。输出张量首位置存储的是输入张量中的最大值，第二个位置存储的是最大值对应的下标。

参数说明

- dst_data: 输出张量，类型: Tensor。
- src_data: 源操作数，类型: Tensor。

返回值描述

- 无返回值。

注意事项

- 输入和输出张量必须定义在 NRAM 空间上。
- 支持的数值类型: float16, float32。
- 输入输出张量元素个数必须是 64 的倍数。

示例

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.amax(Output, Input)
```

7.3.7.5 amin

amin(

dst_data,

src_data

)

选取输入张量中所有元素的最小值，并记录最小值的坐标。输出张量首位置存储的是输入张量中的最小值，第二个位置存储的是最小值对应的下标。

参数说明

- dst_data: 输出张量，类型: Tensor。
- src_data: 源操作数，类型: Tensor。

返回值描述

- 无返回值。

注意事项

- 输入和输出张量必须定义在 NRAM 空间上。
- 支持的数值类型: float16, float32。
- 输入输出张量元素个数必须是 64 的倍数。

示例

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.amin(Output, Input)
```

7.3.7.6 count_nonzero

count_nonzero(

dst_data,

src_data,

bit_count=False

)

统计输入张量里的所有非零元素个数。当 bit_count 为 True 时，统计输入张量里的所有元素的比特位中非零元素个数。

参数说明

- dst_data: 目的操作数，必须定义 NRAM 空间上，类型: Tensor。
- src_data: 源操作数，scope 必须定义 NRAM 空间上，类型: Tensor。
- bit_count: 模式选择参数，类型：Bool。

返回值描述

- 无返回值。

注意事项

- src_data 支持的数值类型有: float16, float32，输出的数值类型为: uint32。
- 输入输出张量元素个数必须是 64 的倍数。

示例

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.count_nonzero(Output, Input)
```

7.3.8 原子运算算子

7.3.8.1 atomic_add

atomic_add(

output,

input1,

input2

)

对输入数据进行原子加法操作。将 input1 与 input2 进行加法运算，运算结果写入 input1，input1 原始的数据写入 output。所有的操作会在一条指令中完成。

参数说明

- output: 输出张量，存储源操作数 input1 在进行加法运算之前的数据，必须位于 NRAM 空间。
- input1: 源操作数 1，类型是张量，与源操作数 2 进行加法运算并存储运算结果，必须位于 GDRAM 空间。
- input2: 源操作数 2，类型可以是张量或标量，必须位于 NRAM 空间。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: uint16, int16, uint32, int32, float16, float32。
- 输入输出张量中的元素个数必须是 16 的倍数，且输出输出张量形状相同，数值类型相同。
- 当 input2 的类型为标量时，input1 张量的元素个数必须为 1。

示例

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="global")
Input2 = bp.Tensor(shape=(a,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.atomic_add(Output, Input1, Input2)
```

7.3.8.2 atomic_and

atomic_and(

output,

input1,

input2

)

对输入数据进行原子相与操作。将 input1 与 input2 进行相与运算，运算结果写入 input1，input1 原始的数据写入 output。所有的操作会在一条指令中完成。

参数说明

- output: 输出张量，存储源操作数 input1 在进行与运算之前的数据，必须位于 NRAM 空间。
- input1: 源操作数 1，类型是张量，与源操作数 2 进行相与运算并存储运算结果，必须位于 GDRAM 空间。
- input2: 源操作数 2，类型可以是张量或标量，必须位于 NRAM 空间。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: uint16, int16, uint32, int32。
- 输入输出张量中的元素个数必须是 16 的倍数，且输出输出张量形状相同，数值类型相同。
- 当 input2 的类型为标量时，input1 张量的元素个数必须为 1。

示例

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.int32, scope="global")
Input2 = bp.Tensor(shape=(a,b), name='Input2', dtype=bangpy.int32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.int32, scope="nram")

bp.atomic_and(Output, Input1, Input2)
```

7.3.8.3 atomic_dec

atomic_dec(

output,

input1,

input2

)

对输入数据进行原子递减运算。将 input1 与 input2 进行递减运算，运算结果写入 input1，input1 原始的数据写入 output。所有的操作会在一条指令中完成。

递减运算: $output = (input1 == 0 || input1 > input2) ? input2 : (input1 - 1)$

参数说明

- output: 输出张量，存储源操作数 input1 在进行递减运算之前的数据，必须位于 NRAM 空间。
- input1: 源操作数 1，类型是张量，与源操作数 2 进行递减运算并存储运算结果，必须位于 GDRAM 空间。
- input2: 源操作数 2，类型可以是张量或标量，必须位于 NRAM 空间。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: uint16, int16, uint32, int32。
- 输入输出张量中的元素个数必须是 16 的倍数，且输出输出张量形状相同，数值类型相同。

- 当 input2 的类型为标量时，input1 张量的元素个数必须为 1。

示例

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.int32, scope="global")
Input2 = bp.Tensor(shape=(a,b), name='Input2', dtype=bangpy.int32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.int32, scope="nram")

bp.atomic_dec(Output, Input1, Input2)
```

7.3.8.4 atomic_exch

atomic_exch(

output,

input1,

input2

)

对输入数据进行原子交换运算。将 input2 的数据写入 input1，input1 原始的数据写入 output。所有的操作会在一条指令中完成。

参数说明

- output: 输出张量，存储源操作数 input1 在进行交换运算之前的数据，必须位于 NRAM 空间。
- input1: 源操作数 1，类型是张量，与源操作数 2 进行交换运算并存储运算结果，必须位于 GDRAM 空间。
- input2: 源操作数 2，类型可以是张量或标量，必须位于 NRAM 空间。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: uint16, int16, uint32, int32，float16, float32。
- 输入输出张量中的元素个数必须是 16 的倍数，且输出输出张量形状相同，数值类型相同。
- 当 input2 的类型为标量时，input1 张量的元素个数必须为 1。

示例

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.float32, scope="global")
Input2 = bp.Tensor(shape=(a,b), name='Input2', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.float32, scope="nram")

bp.atomic_exch(Output, Input1, Input2)
```

7.3.8.5 atomic_inc

atomic_inc(

output,

input1,

input2

)

对输入数据进行原子递加运算。将 input1 与 input2 进行递加运算，运算结果写入 input1，input1 原始的数据写入 output。所有的操作会在一条指令中完成。

递减运算: $output = (input1 >= input2)?0 : (input1 + 1)$

参数说明

- output: 输出张量，存储源操作数 input1 在进行递加运算之前的数据，必须位于 NRAM 空间。
- input1: 源操作数 1，类型是张量，与源操作数 2 进行递加运算并存储运算结果，必须位于 GDRAM 空间。
- input2: 源操作数 2，类型可以是张量或标量，必须位于 NRAM 空间。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: uint16, int16, uint32, int32。
- 输入输出张量中的元素个数必须是 16 的倍数，且输出输出张量形状相同，数值类型相同。
- 当 input2 的类型为标量时，input1 张量的元素个数必须为 1。

示例

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.int32, scope="global")
Input2 = bp.Tensor(shape=(a,b), name='Input2', dtype=bangpy.int32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.int32, scope="nram")

bp.atomic_inc(Output, Input1, Input2)
```

7.3.8.6 atomic_max

atomic_max(

output,

input1,

input2

)

将 input1 与 input2 进行逐元素比较运算并将较大值写入 input1，input1 原始的数据写入 output。所有的操作会在一条指令中完成。

参数说明

- output: 输出张量，存储源操作数 input1 在进行比较运算之前的数据，必须位于 NRAM 空间。
- input1: 源操作数 1，类型是张量，与源操作数 2 进行比较运算并存储运算结果，必须位于 GDRAM 空间。
- input2: 源操作数 2，类型可以是张量或标量，必须位于 NRAM 空间。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: uint16, int16, uint32, int32, float16, float32。
- 输入输出张量中的元素个数必须是 16 的倍数，且输出输出张量形状相同，数值类型相同。
- 当 input2 的类型为标量时，input1 张量的元素个数必须为 1。

示例

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.int32, scope="global")
Input2 = bp.Tensor(shape=(a,b), name='Input2', dtype=bangpy.int32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.int32, scope="nram")

bp.atomic_max(Output, Input1, Input2)
```

7.3.8.7 atomic_min

atomic_min(

output,

input1,

input2

)

将 input1 与 input2 进行逐元素比较运算并将较小值写入 input1，input1 原始的数据写入 output。所有的操作会在一条指令中完成。

参数说明

- output: 输出张量，存储源操作数 input1 在进行比较运算之前的数据，必须位于 NRAM 空间。
- input1: 源操作数 1，类型是张量，与源操作数 2 进行比较运算并存储运算结果，必须位于 GDRAM 空间。
- input2: 源操作数 2，类型可以是张量或标量，必须位于 NRAM 空间。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: uint16, int16, uint32, int32, float16, float32。
- 输入输出张量中的元素个数必须是 16 的倍数，且输出输出张量形状相同，数值类型相同。
- 当 input2 的类型为标量时，input1 张量的元素个数必须为 1。

示例

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.int32, scope="global")
Input2 = bp.Tensor(shape=(a,b), name='Input2', dtype=bangpy.int32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.int32, scope="nram")

bp.atomic_min(Output, Input1, Input2)
```


7.3.8.8 atomic_or

atomic_or(

output,

input1,

input2

)

对输入数据进行原子相或操作。将 input1 与 input2 进行相或运算，运算结果写入 input1，input1 原始的数据写入 output。所有的操作会在一条指令中完成。

参数说明

- output: 输出张量，存储源操作数 input1 在进行或运算之前的数据，必须位于 NRAM 空间。

- input1: 源操作数 1，类型是张量，与源操作数 2 进行相或运算并存储运算结果，必须位于 GDRAM 空间。
- input2: 源操作数 2，类型可以是张量或标量，必须位于 NRAM 空间。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: uint16, int16, uint32, int32。
- 输入输出张量中的元素个数必须是 16 的倍数，且输出输出张量形状相同，数值类型相同。
- 当 input2 的类型为标量时，input1 张量的元素个数必须为 1。

示例

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.int32, scope="global")
Input2 = bp.Tensor(shape=(a,b), name='Input2', dtype=bangpy.int32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.int32, scope="nram")

bp.atomic_or(Output, Input1, Input2)
```

7.3.8.9 atomic_xor

atomic_xor(
output,
input1,
input2
)

对输入数据进行原子异或操作。将 input1 与 input2 进行异或运算，运算结果写入 input1，input1 原始的数据写入 output。所有的操作会在一条指令中完成。

参数说明

- output: 输出张量，存储源操作数 input1 在进行异或运算之前的数据，必须位于 NRAM 空间。
- input1: 源操作数 1，类型是张量，与源操作数 2 进行异或运算并存储运算结果，必须位于 GDRAM 空间。
- input2: 源操作数 2，类型可以是张量或标量，必须位于 NRAM 空间。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: uint16, int16, uint32, int32。
- 输入输出张量中的元素个数必须是 16 的倍数，且输出输出张量形状相同，数值类型相同。
- 当 input2 的类型为标量时，input1 张量的元素个数必须为 1。

示例

```
Input1 = bp.Tensor(shape=(a,b), name='Input1', dtype=bangpy.int32, scope="global")
Input2 = bp.Tensor(shape=(a,b), name='Input2', dtype=bangpy.int32, scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype=bangpy.int32, scope="nram")

bp.atomic_xor(Output, Input1, Input2)
```

7.3.9 矩阵运算算子

7.3.9.1 conv

conv(

dst_data,

src_data,

kernel,

stride_x,

stride_y,

fix_postion,

bias=None

)

卷积算子。

参数说明

- dst_data: 目的操作数，必须定义在 NRAM 空间上，类型: Tensor。
- src_data: 源操作数，必须定义在 NRAM 空间上，类型: Tensor。
- kernel: 卷积核，必须定义在 WRAM 空间上，类型: Tensor。
- stride_x : 沿 x 方向的 stride，类型: int。
- stride_y : 沿 y 方向的 stride，类型: int。
- fix_postion: 量化参数，范围为 [-127, 127], 类型: int.
- bias: 偏置，当声明成 Tensor 时，必须定义在 NRAM 上，类型: Tensor or None。

返回值描述

- 无返回值。

注意事项

- 源操作数的 shape 为 [in, ih, iw, ic]
- 卷积核的 shape 为 [oc, kh, kw, ic]
- 偏置的大小为输出通道数的大小。
- 输出张量的 shape 为 [oc, oh, ow, in]。参数计算如下:

  $oh = (ih - kh)/stride\_x + 1$

  $ow = (iw - kw)/stride\_y + 1$
- 约束限制:

  1) 输入通道数长度必须 64 字节对齐。

  2) 输出通道数长度必须 128 字节对齐。
- 支持的数据类型如下表所示:

表 7.1: conv 算子参数支持的数据类型

src_data	kernel	dst_data	bias
int8	int8	float16	None
int8	int8	float32	None
int8	int8	int16	None
int16	int8	float16	None
int16	int8	float32	None
int16	int8	int16	None
int16	int16	float16	None
int16	int16	float32	None
int8	int8	float16	float16
int8	int8	float32	float32
int8	int8	int16	int16
int16	int8	float16	float16
int16	int8	float32	float32
int16	int8	int16	int16
int16	int16	float16	float16
int16	int16	float32	float32

示例

```
Input = bp.Tensor(shape=(in,ih,iw,ic), name='Input', dtype=bangpy.int8, scope="nram")
kernel = bp.Tensor(shape=(oc,kh,kw,kc), name='kernel', dtype=bangpy.int8, scope="wram")
bias = bp.Tensor(shape=(oc,), name='bias', dtype=bangpy.float16, scope="nram")
Output = bp.Tensor(shape=(oc,oh,ow,in), name='Output', dtype=bangpy.float16, scope="nram")

bp.conv(Output, Input, kernel, stride_x, stride_y, 0, bias)
```


7.3.9.2 conv_partial

```
conv_partial(
    dst_data,
    src_data,
        kernel,
        partial,
        stride_x,
        stride_y,
        fix_postion
)
```

卷积算子。

参数说明

- dst_data: 目的操作数，必须定义在 NRAM 上，类型: Tensor。
- src_data: 源操作数，必须定义在 NRAM 上，类型: Tensor。
- kernel: 卷积核，必须定义在 WRAM 上，类型: Tensor。
- partial: 偏置，必须定义在 NRAM 上，类型: Tensor。
- stride_x : 沿 x 方向的 stride，类型: int。
- stride_y : 沿 y 方向的 stride，类型: int。
- fix_postion: 量化参数，范围为 [-127, 127], 类型: int.

返回值描述

- 无返回值。

注意事项

- 源操作数的 shape 为 [in, ih, iw, ic]
- keinel 的 shape 为 [oc, kh, kw, ic]
- partial 的大小为输出张量的 shape 一致。
- 输出张量的 shape 为 [oc, oh, ow, in]。参数计算如下:

  $oh = (ih - kh)/stride\_x + 1$

  $ow = (iw - kw)/stride\_y + 1$
- 约束限制:

  1) 输入通道数长度必须 64 字节对齐。

  2) 输出通道数长度必须 128 字节对齐。
- 支持的数据类型如下表所示:

表 7.2: conv_partial 算子参数支持的数据类型

src_data	kernel	dst_data	partial
int8	int8	float32	float32
int16	int8	float32	float32
int16	int16	float32	float32
int16	int16	float16	float16
int16	int8	float16	float16
int8	int8	float16	float16

示例

```
Input = bp.Tensor(shape=(in,ih,iw,ic), name='Input', dtype=bangpy.int8, scope="nram")
kernel = bp.Tensor(shape=(oc,kh,kw,kc), name='kernel', dtype=bangpy.int8, scope="wram")
partial = bp.Tensor(shape=(oc,oh,ow,oc), name='partial', dtype=bangpy.float16, scope="nram")
Output = bp.Tensor(shape=(oc,oh,ow,in), name='Output', dtype=bangpy.float16, scope="nram")

bp.conv_partial(Output, Input, kernel, partial, stride_x, stride_y, 0)
```


7.3.9.3 dense

dense(

dst_data,

src_data,

weight,

fix_postion,

bias=None

)

对源操作数进行线性变换，数学变换可描述为:

有 bias，$dst\_data = src\_data \cdot weight + bias$。

无 bias，$dst\_data = src\_data \cdot weight$

参数说明

- dst_data: 目的操作数，数据格式 [H, W]，必须位于 NRAM 空间上，类型: Tensor。
- src_data: 源操作数, 数据格式 [H, W]，必须位于 NRAM 空间上，类型: Tensor。
- weight: 权重，数据格式 [H,W]，必须位于 WRAM 空间上，类型: Tensor。
- fix_postion: 立即数，范围是 [-127,127]，类型:int。
- bias: 偏置，当为 Tensor 时，shape 与输入张量和权重进行矩阵乘后输出的 shape 一致, 必须位于 NRAM 空间上。类型: None or Tensor。

返回值描述

- 无返回值。

注意事项

- 约束限制:

  1) weight 的 height 必须被 64 整除，weight 的 width 长度必须 64 字节对齐。
- 支持的数据类型如下表所示:

表 7.3: dense 算子参数支持的数据类型

dst_data	src_data	weight	bias
float16	int16	int16	float16
float16	int8	int8	float16
float16	int16	int8	float16
float32	int16	int16	float32
float32	int8	int8	float32
float32	int16	int8	float32
float32	int32	int16	float32
int16	int16	int16	int16
int16	int8	int8	int16
int16	int16	int8	int16

示例

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype=bangpy.int8, scope="nram")
weight = bp.Tensor(shape=(b, c), name='weight', dtype=bangpy.int8, scope="wram")
bias = bp.Tensor(shape=(a, c), name='bias', dtype=bangpy.float16, scope="nram")
```

```
Output = bp.Tensor(shape=(a+1+2,b+3+4,c), name='Output', dtype=bangpy.float16, scope="nram")

bp.dense(Output, Input, weight, 0, bias)
```

7.3.10 数据填充算子

7.3.10.1 pad

pad(

dst_data,

src_data,

pad_top,

pad_bottom,

pad_left,

pad_right

)

对源操作数的 H 和 W 进行 0 填充，并将填充结果存在目的操作数中。

参数说明

- dst_data: 目的操作数，数据格式 [H, W, C]，必须位于 NRAM 空间上，类型: Tensor。
- src_data: 源操作数, 数据格式 [H, W, C]，必须位于 NRAM 空间上，类型: Tensor。
- pad_top: 在源操作数的 height 的上方填 0，类型: int
- pad_bottom: 在源操作数 height 的下方填 0，类型:int。
- pad_left: 在源操作数 width 的左边填 0，类型: int。
- pad_right: 在源操作数 width 的右边填 0，类型: int。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, float32, int32。
- 目的操作数参数计算:

  $dst\_height = (src\_height + pad\_top + pad\_bottom)$

  $dst\_width = src\_width + pad\_left + pad\_right$
- 约束限制:

  1) $(channel \times src\_width) \div 64 == 0$

  2) $((pad\_left + pad\_right) \times channel \times sizeof(type)) \div 128 == 0$

3) $((pad\_top \times (pad\_left + pad\_right + src\_width) + pad\_left) \times channel) \div 64 == 0$

4) $((pad\_bottom \times (pad\_left + pad\_right + width) + pad\_right) \times channel) \div 64 == 0$

示例

```
Input = bp.Tensor(shape=(a,b,c), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(a+1+2,b+3+4,c), name='Output', dtype=bangpy.float32, scope="nram")

bp.pad(Output, Input, 1, 2, 3, 4)
```

7.3.11 池化算子

7.3.11.1 avgpool

avgpool(

dst_data,

src_data,

kernel_height,

kernel_width,

stride_x,

stride_y

)

平均池化操作。

参数说明

- dst_data: 取滑动窗口内所有元素和的平均值，是一个 shape=(h,w,c) 的三维张量，类型: Tensor。
- src_data: 源操作数，是一个 shape=(h,w,c) 的三维张量，类型: Tensor。
- kernel_height: 滑动窗口的高度，类型: int。
- kernel_width: 滑动窗口的宽度，类型: int。
- stride_x: 沿 x 方向的步长，类型: int。
- stride_y: 沿 y 方向的步长，类型: int。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, float32。
- 通道数 (c) 必须 128 字节对齐。

示例

```
Input = bp.Tensor(shape=(h,w,c), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(oh,ow,oc), name='Output', dtype=bangpy.float32, scope="nram")

bp.avgpool(Output, Input, kernel_height, kernel_width, stride_x, stride_y)
```

7.3.11.2 avgpool_bp

avgpool_bp(

dst_data,

src_data,

kernel_height,

kernel_width,

stride_x,

stride_y

)

反向平均池化操作。

参数说明

- dst_data: 目的操作数，是一个 shape=(h,w,c) 的三维张量，类型: Tensor。
- src_data: 源操作数，是一个 shape=(h,w,c) 的三维张量，类型: Tensor。
- kernel_height: 滑动窗口的高度，类型: int。
- kernel_width: 滑动窗口的宽度，类型: int。
- stride_x: 沿 x 方向的步长，类型: int。
- stride_y: 沿 y 方向的步长，类型: int。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, float32。
- 通道数 (c) 必须 128 字节对齐。
- 滑动窗口的高度 kernel_height 必须大于等于沿 x 方向的步长 stride_x，滑动窗口的宽度 kernel_width 必须大于等于沿 y 方向的步长 stride_y。

示例

```
Input = bp.Tensor(shape=(h,w,c), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(oh,ow,oc), name='Output', dtype=bangpy.float32, scope="nram")

bp.avgpool_bp(Output, Input, kernel_height, kernel_width, stride_x, stride_y)
```

7.3.11.3 maxpool

maxpool(

dst_data,

src_data,

kernel_height,

kernel_width,

stride_x,

stride_y

)

最大池化操作。

参数说明

- dst_data: 取滑动窗口内的最大值，是一个 shape=(h,w,c) 的三维矩阵，类型: Tensor。
- src_data: 源操作数，是一个 shape=(h,w,c) 的三维矩阵，类型: Tensor。
- kernel_height: 滑动窗口的高度，类型: int。
- kernel_width: 滑动窗口的宽度，类型: int。
- stride_x: 沿 x 方向的步长，类型: int。
- stride_y: 沿 y 方向的步长，类型: int。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, float32。
- 通道数 (c) 必须 128 字节对齐。

示例

```
Input = bp.Tensor(shape=(h,w,c), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(oh,ow,oc), name='Output', dtype=bangpy.float32, scope="nram")

bp.maxpool(Output, Input, kernel_height, kernel_width, stride_x, stride_y)
```


7.3.11.4 maxpool_index

maxpool_index(

dst_data,

src_data,

kernel_height,

kernel_width,

stride_x,

stride_y

)

最大池化操作过程中的位置标记。

参数说明

- dst_data: 取滑动窗口内最大值的标记位置，是一个 shape=(h,w,c) 的三维矩阵，类型: Tensor。
- src_data: 源操作数，是一个 shape=(h,w,c) 的三维矩阵，类型: Tensor。
- kernel_height: 滑动窗口的高度，类型: int。
- kernel_width: 滑动窗口的宽度，类型: int。
- stride_x: 沿 x 方向的步长，类型: int。
- stride_y: 沿 y 方向的步长，类型: int。

返回值描述

- 无返回值。

注意事项

- src_data 中支持的数值类型: float16, float32，对应 dst_data 中数值类型为 uint16, uint32。
- 通道数 (c) 必须 128 字节对齐。

示例

```
Input = bp.Tensor(shape=(h,w,c), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(oh,ow,oc), name='Output', dtype=bangpy.float32, scope="nram")

bp.maxpool_index(Output, Input, kernel_height, kernel_width, stride_x, stride_y)
```

7.3.11.5 maxpool_bp

maxpool_bp(

dst_data,

src_data,

kernel_height,

kernel_width,

stride_x,

stride_y

)

反向最大池化操作。

参数说明

- dst_data: 目的操作数，是一个 shape=(h,w,c) 的三维张量，类型: Tensor。
- src_data: 源操作数，是一个 shape=(h,w,c) 的三维张量，类型: Tensor。
- kernel_height: 滑动窗口的高度，类型: int。
- kernel_width: 滑动窗口的宽度，类型: int。
- stride_x: 沿 x 方向的步长，类型: int。
- stride_y: 沿 y 方向的步长，类型: int。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, float32。
- 通道数 (c) 必须 128 字节对齐。
- 滑动窗口的高度 kernel_height 必须大于等于沿 x 方向的步长 stride_x，滑动窗口的宽度 kernel_width 必须大于等于沿 y 方向的步长 stride_y。

示例

```
Input = bp.Tensor(shape=(h,w,c), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(oh,ow,oc), name='Output', dtype=bangpy.float32, scope="nram")

bp.avgpool_bp(Output, Input, kernel_height, kernel_width, stride_x, stride_y)
```

7.3.11.6 minpool

minpool(

dst_data,

src_data,

kernel_height,

kernel_width,

stride_x,

stride_y

)

最小池化操作。

参数说明

- dst_data: 取滑动窗口内的最小值，是一个 shape=(h,w,c) 的三维张量，类型: Tensor。
- src_data: 源操作数，是一个 shape=(h,w,c) 的三维张量，类型: Tensor。
- kernel_height: 滑动窗口的高度，类型: int。
- kernel_width: 滑动窗口的宽度，类型: int。
- stride_x: 沿 x 方向的步长，类型: int。
- stride_y: 沿 y 方向的步长，类型: int。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, float32。
- 通道数 (c) 必须 128 字节对齐。

示例

```
Input = bp.Tensor(shape=(h,w,c), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(oh,ow,oc), name='Output', dtype=bangpy.float32, scope="nram")

bp.minpool(Output, Input, kernel_height, kernel_width, stride_x, stride_y)
```

7.3.11.7 minpool_index

minpool_index(

dst_data,

src_data,

kernel_height,

kernel_width,

stride_x,

stride_y

)

最小池化操作过程中的位置标记。

参数说明

- dst_data: 取滑动窗口内最小值的标记位置，是一个 shape=(h,w,c) 的三维矩阵，类型: Tensor。
- src_data: 源操作数，是一个 shape=(h,w,c) 的三维矩阵，类型: Tensor。
- kernel_height: 滑动窗口的高度，类型: int。
- kernel_width: 滑动窗口的宽度，类型: int。
- stride_x: 沿 x 方向的步长，类型: int。
- stride_y: 沿 y 方向的步长，类型: int。

返回值描述

- 无返回值。

注意事项

- src_data 中支持的数值类型: float16, float32，对应 dst_data 中数值类型为 uint16, uint32。
- 通道数 (c) 必须 128 字节对齐。

示例

```
Input = bp.Tensor(shape=(h,w,c), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(oh,ow,oc), name='Output', dtype=bangpy.float32, scope="nram")

bp.minpool_index(Output, Input, kernel_height, kernel_width, stride_x, stride_y)
```

7.3.11.8 sumpool

sumpool(

dst_data,

src_data,

kernel_height,

kernel_width,

stride_x,

stride_y

)

加和池化操作。

参数说明

- dst_data: 取滑动窗口内的所有元素的和，是一个 shape=(h,w,c) 的三维张量，类型: Tensor。
- src_data: 源操作数，是一个 shape=(h,w,c) 的三维张量，类型: Tensor。
- kernel_height: 滑动窗口的高度，类型: int。
- kernel_width: 滑动窗口的宽度，类型: int。
- stride_x: 沿 x 方向的步长，类型: int。
- stride_y: 沿 y 方向的步长，类型: int。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, float32。
- 通道数 (c) 必须 128 字节对齐。

示例

```
Input = bp.Tensor(shape=(h,w,c), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(oh,ow,oc), name='Output', dtype=bangpy.float32, scope="nram")

bp.sumpool(Output, Input, kernel_height, kernel_width, stride_x, stride_y)
```

7.3.11.9 unpool

```
unpool(
    dst_data,
    src_data,
    kernel_height,
    kernel_width,
    stride_x,
    stride_y,
    index
)
```

上采样算子。

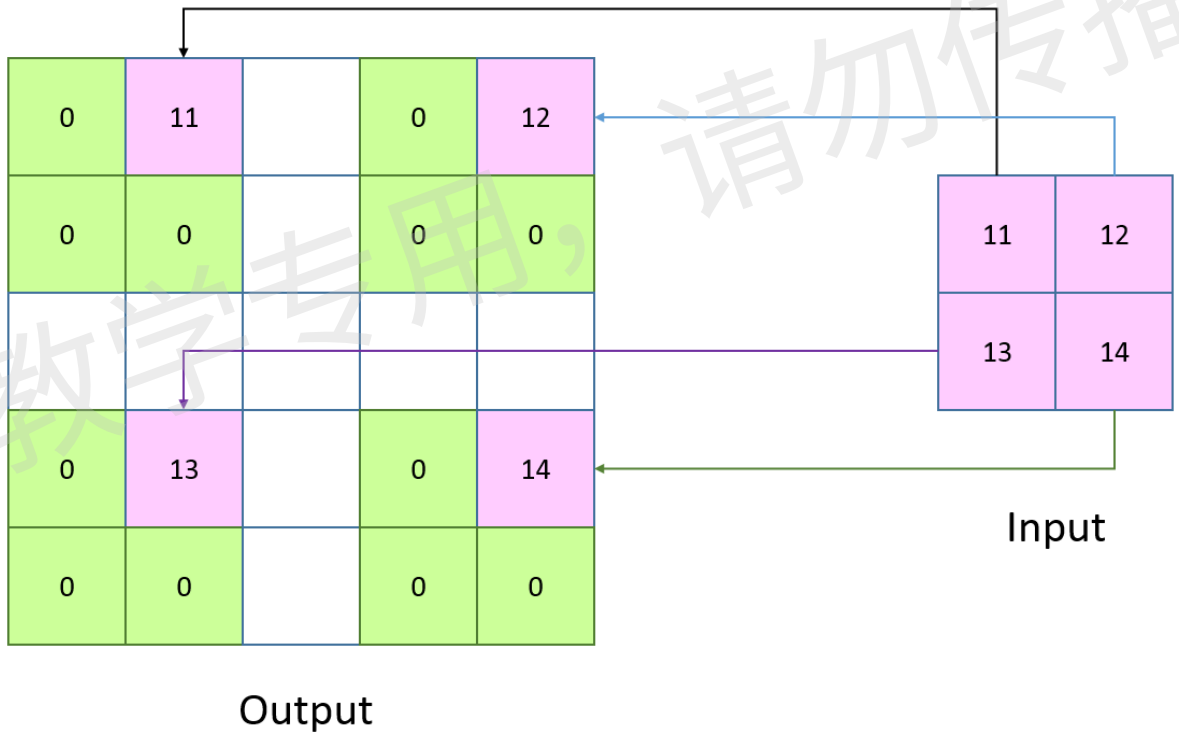

图 7.1: unpool 示意图

参数说明

- dst_data: 取滑动窗口内的最小值，是一个 shape=(h,w,c) 的三维张量，类型: Tensor。
- src_data: 源操作数，是一个 shape=(h,w,c) 的三维张量，类型: Tensor。
- kernel_height: 滑动窗口的高度，类型: int。
- kernel_width: 滑动窗口的宽度，类型: int。

- stride_x: 沿 x 方向的步长，类型: int。
- stride_y: 沿 y 方向的步长，类型: int。
- index: 反向池化映射回滑动窗口的元素位置坐标，类型: int。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, float32。
- 通道数 (c) 必须是 64 的倍数。
- 滑动窗口的高度 kernel_height 必须大于等于沿 x 方向的步长 stride_x，滑动窗口的宽度 kernel_width 必须大于等于沿 y 方向的步长 stride_y。
- index 的大小必须小于滑动窗口的高度 kernel_height 与滑动窗口 kernel_width 的宽度的乘积。

示例

```
Input = bp.Tensor(shape=(h,w,c), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(oh,ow,oc), name='Output', dtype=bangpy.float32, scope="nram")

bp.unpool(Output, Input, kernel_height, kernel_width, stride_x, stride_y, index)
```

7.3.12 形状变换算子

7.3.12.1 transpose

transpose(

dst_data,

src_data,

axes=None

)

对张量进行形状变换。

参数说明

- dst_data: 形状变换后的数据，类型: Tensor，数值类型与源操作数相同。
- src_data: 源操作数，类型: Tensor。
- axes: 张量维度轴的变换顺序，类型: tuple、list。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, int16, uint16, int8, float32, int32, uint32。
- 当 axes 为 None 时，表示该操作是对二维张量的转置，src_data 与 dst_data 必须是二维张量，且每个维度上的数据长度必须 64 字节对齐。当 axes 不为 None 时，表示该操作是对四维张量 (N,H,W,C) 的形状变换。
- 四维张量的 H 和 W 必须 64 字节对齐，C 必须被 64 整除，目前支持的 axes 输入有 (0,3,1,2) 与 (0,2,3,1)。

示例

```
Input = bp.Tensor(shape=(n,h,w,c), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(n,c,h,w), name='Output', dtype=bangpy.float32, scope="nram")

bp.transpose(Output, Input, axes=(0,3,1,2))
```

7.3.12.2 fliplr

fliplr(

dst_data,

src_data

)

对二维张量的数据在行方向上进行镜像翻转。

参数说明

- dst_data: 翻转后的数据，类型: Tensor，数值类型与源操作数相同。
- src_data: 源操作数，是一个 shape=(h,w) 的二维张量，类型: Tensor。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, int16, uint16, int8, float32, int32, uint32。
- 输入输出张量的 H 和 W 必须是 16 的倍数且输入输出张量的形状相同。

示例

```
Input = bp.Tensor(shape=(n,h,w,c), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(n,c,h,w), name='Output', dtype=bangpy.float32, scope="nram")

bp.fliplr(Output, Input)
```


7.3.12.3 rot90

rot90(

dst_data,

src_data

)

将二维张量按顺时针旋转 90 度。

参数说明

- dst_data: 旋转后的数据，类型: Tensor，数值类型与源操作数相同。
- src_data: 源操作数，是一个 shape=(h,w) 的二维张量，类型: Tensor。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, int16, uint16, int8, float32, int32, uint32。
- 输入输出张量的 H 和 W 必须 128 字节对齐且输入输出张量的长度相同。

示例

```
Input = bp.Tensor(shape=(n,h,w,c), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(n,c,h,w), name='Output', dtype=bangpy.float32, scope="nram")

bp.rot90(Output, Input)
```

7.3.12.4 rot180

rot180(

dst_data,

src_data

)

将二维张量按顺时针旋转 180 度。

参数说明

- dst_data: 旋转后的数据，类型: Tensor，数值类型与源操作数相同。
- src_data: 源操作数，是一个 shape=(h,w) 的二维张量，类型: Tensor。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, int16, uint16, int8, float32, int32, uint32。
- 输入输出张量的 H 和 W 必须 128 字节对齐且输入输出张量的长度相同。

示例

```
Input = bp.Tensor(shape=(n,h,w,c), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(n,c,h,w), name='Output', dtype=bangpy.float32, scope="nram")

bp.rot180(Output, Input)
```

7.3.12.5 rot270

rot270(

dst_data,

src_data

)

将二维张量按顺时针旋转 270 度。

参数说明

- dst_data: 旋转后的数据，类型: Tensor，数值类型与源操作数相同。
- src_data: 源操作数，是一个 shape=(h,w) 的二维张量，类型: Tensor。

返回值描述

- 无返回值。

注意事项

- 支持的数值类型: float16, int16, uint16, int8, float32, int32, uint32。
- 输入输出张量的 H 和 W 必须 128 字节对齐且输入输出张量的长度相同。

示例

```
Input = bp.Tensor(shape=(n,h,w,c), name='Input', dtype=bangpy.float32, scope="nram")
Output = bp.Tensor(shape=(n,c,h,w), name='Output', dtype=bangpy.float32, scope="nram")

bp.rot270(Output, Input)
```

7.3.13 张量数值类型转换算子

下文提到的舍入模式，如下表所示：

表 7.4: 舍入模式示意表

舍入类型	描述	原始数据	舍入后的数据
tz	向零取整	1.5	1
tz	向零取整	1.4	1
tz	向零取整	-1.5	-1
oz	远离零取整	1.5	2
oz	远离零取整	1.4	2
oz	远离零取整	-1.5	-2
rd	四舍五入	1.5	2
rd	四舍五入	1.4	1
rd	四舍五入	-1.5	-2
dn	向下取整	1.5	1
dn	向下取整	1.4	1
dn	向下取整	-1.5	-2
up	向上取整	1.5	2
up	向上取整	1.4	2
up	向上取整	-1.5	-1

7.3.13.1 type_convert

type_convert(

dst_data,

src_data,

fix_pos,

rounding=" rd"

)

转换输入数值类型。

参数说明

- dst_data: 转换之后的数据，类型: Tensor。
- src_data: 要被转换的数据，类型: Tensor。
- fix_pos: 缩放参数，类型: int。
- rounding: 舍入类型，包含有: rd: 四舍五入; dn: 向下取整; up: 向上取整; oz: 远离 0 取整; tz: 向 0 取整，类型：str。

返回值描述

- 无返回值。

注意事项

- src_data 的数据长度必须 128 字节对齐。
- fix_pos 的范围为 [-127,127]，当数据转换是从高精度转低精度时，如 float32 转 int16/int8，float16 转 int16，$dst = src/2^{fix\_pos}$。当数据转换时从低精度转高精度时，如 int16 转 float32/float16，int8 转 float32，$dst = src * 2^{fix\_pos}$。

示例

```
Input = bp.Tensor(shape=(a,b), name='Input', dtype="int16", scope="nram")
Output = bp.Tensor(shape=(a,b), name='Output', dtype="float16", scope="nram")

bp.type_convert(Output, Input, fix_pos=1)
```

7.4 标量计算接口

7.4.1 scalar_max

scalar_max(

value_a,

value_b

)

返回两个输入标量的最大值。

参数说明

- value_a、value_b: 输入标量，类型：Scalar。

返回值描述

- 返回两个输入标量的最大值，类型：Scalar。

注意事项

- 支持的数值类型：int32，float32。

示例

```
Input1 = bp.Scalar(name='Input1', dtype=bangpy.float32, value=6)
Input2 = bp.Scalar(name='Input2', dtype=bangpy.float32, value=6)

print(bp.scalar_max(Input1, Input2))
```

7.4.2 scalar_min

scalar_min(

value_a,

value_b

)

返回两个输入标量的最小值。

参数说明

- value_a、value_b: 输入标量，类型：Scalar。

返回值描述

- 返回两个输入标量的最小值，类型：Scalar。

注意事项

- 支持的数值类型：int32，float32。

示例

```
Input1 = bp.Scalar(name='Input1', dtype=bangpy.float32, value=6)
Input2 = bp.Scalar(name='Input2', dtype=bangpy.float32, value=6)

print(bp.scalar_min(Input1, Input2))
```

7.4.3 scalar_pow

scalar_pow(

value_a,

value_b

)

返回底数为 value_a，基数为 value_b 的幂运算的值。

参数说明

- value_a、value_b: 输入标量，类型：Scalar。

返回值描述

- 返回两个输入标量的最大值，类型：Scalar。

注意事项

- 支持的数值类型：float32。

示例

```
Input1 = bp.Scalar(name='Input1', dtype=bangpy.float32, value=6)
Input2 = bp.Scalar(name='Input2', dtype=bangpy.float32, value=6)

print(bp.scalar_pow(Input1, Input2))
```

7.4.4 scalar_abs

scalar_abs(

src_data

)

返回输入标量的绝对值。

参数说明

- src_data: 输入标量，类型：Scalar。

返回值描述

- 返回输入标量的绝对值，类型：Scalar。

注意事项

- 支持的数值类型：int16，int32, float32。

示例

```
Input1 = bp.Scalar(name='Input1', dtype=bangpy.int32, value=6)

print(bp.scalar_abs(Input1))
```

7.4.5 scalar_sin

scalar_sin(

src_data

)

返回输入标量的正弦值。

参数说明

- src_data: 输入标量，类型：Scalar。

返回值描述

- 返回输入标量的正弦值，类型：Scalar。

注意事项

- 支持的数值类型：float32。

示例

```
Input1 = bp.Scalar(name='Input1', dtype=bangpy.float32, value=6)

print(bp.scalar_sin(Input1))
```

7.4.6 scalar_cos

scalar_cos(

src_data

)

返回输入标量的余弦值。

参数说明

- src_data: 输入标量，类型：Scalar。

返回值描述

- 返回输入标量的余弦值，类型：Scalar。

注意事项

- 支持的数值类型：float32。

示例

```
Input1 = bp.Scalar(name='Input1', dtype=bangpy.float32, value=6)

print(bp.scalar_cos(Input1))
```

7.4.7 scalar_log

scalar_log(

src_data

)

返回输入标量的底数为 e 的对数运算结果。

参数说明

- src_data: 输入标量，类型：Scalar。

返回值描述

- 返回输入标量的底数为 e 的对数运算结果，类型：Scalar。

注意事项

- 支持的数值类型：float32。

示例

```
Input1 = bp.Scalar(name='Input1', dtype=bangpy.float32, value=6)

print(bp.scalar_log(Input1))
```

7.4.8 scalar_sqrt

scalar_sqrt(

src_data

)

对输入标量进行开方运算。

参数说明

- src_data: 输入标量，类型：Scalar。

返回值描述

- 返回输入标量开方运算的结果，类型：Scalar。

注意事项

- 支持的数值类型：float32。

示例

```
Input1 = bp.Scalar(name='Input1', dtype=bangpy.float32, value=6)

print(bp.scalar_sqrt(Input1))
```

7.4.9 scalar_trunc

scalar_trunc(

src_data

)

对输入标量进行舍尾运算。

参数说明

- src_data: 输入标量，类型：Scalar。

返回值描述

- 返回输入标量舍尾运算的结果，类型：Scalar。

注意事项

- 支持的数值类型：float32。

示例

```
Input1 = bp.Scalar(name='Input1', dtype=bangpy.float32, value=6)

print(bp.scalar_trunc(Input1))
```

7.4.10 scalar_ceil

scalar_ceil(

src_data

)

对输入标量向上取整。

参数说明

- src_data: 输入标量，类型：Scalar。

返回值描述

- 返回输入标量向上取整的结果，类型：Scalar。

注意事项

- 支持的数值类型：half，float32。

示例

```
Input1 = bp.Scalar(name='Input1', dtype=bangpy.float32, value=6)

print(bp.scalar_ceil(Input1))
```

7.4.11 scalar_floor

scalar_floor(

src_data

)

对输入标量向下取整。

参数说明

- src_data: 输入标量，类型：Scalar。

返回值描述

- 返回输入标量向下取整的结果，类型：Scalar。

注意事项

- 支持的数值类型：half，float32。

示例

```
Input1 = bp.Scalar(name='Input1', dtype=bangpy.float32, value=6)

print(bp.scalar_floor(Input1))
```

7.4.12 scalar_round

scalar_round(

src_data

)

对输入标量进行四舍五入运算。

参数说明

- src_data: 输入标量，类型：Scalar。

返回值描述

- 返回输入标量四舍五入的结果，类型：Scalar。

注意事项

- 支持的数值类型：half，float32。

示例

```
Input1 = bp.Scalar(name='Input1', dtype=bangpy.float32, value=6)

print(bp.scalar_round(Input1))
```

7.5 控制流接口

7.5.1 for_range

for_range(

begin,

end,

name=”i”,

stage=1,

task_num=1,

task_type=TaskType.BLOCK,

dtype=”int32”

)

创建一个 BANGPy 的 for 循环语句，可在 for 循环中循环执行语句和开启多核运行功能。

参数说明

- begin: 循环起点，类型: Scalar, int。
- end: 循环终点，类型: Scalar, int。
- name: 循环迭代的索引名，若没有输入的名字，则使用典型的下标名 i, j, k 等，类型: str。
- stage: 预留参数，未来支持循环中做流水，用来设置流水的 stage 数, 取值为 [1, 2, 3, 4]。
- task_num: for 循环中发射的任务数，最大值 65535。如果 task_num>1，则在当前 for_range 作用域内的代码可以并行地在不同的 mlu 核上运行，类型: str。
- task_type: 指定在一个 MLU 上能并行运行的任务数量的任务类型，类型: TaskType。
- dtype: 迭代变量的数值类型，类型: str。

返回值描述

- loop_scope: for 语句作用域。

注意事项

- task_num 和 task_type 必须相互匹配。具体地说，task_num 必须能被 task_type 对应的并行任务数整除。

- task_type 有五种取值：TaskType.BLOCK，TaskType.UNION1，UNION2, UNION4。分别对应的并行任务数为 1, 4, 8, 16。
- for_range 在 with…as…中使用，as 的结果是 for 循环的 Scalar 类型循环变量。
- 目前 BANGPy 暂不支持对多个 for 循环设置不同的 task_num 和 task_type 进行多核展开。
- 流水只能在 NRAM 上进行，即 Load(GDRAM-NRAM), compute, store(NRAM->GDRAM) 的程序执行步骤。

示例

```
tcp_container = tcp.TCP()
# Normal for_range usage which will be mapped to the for clause of C/C++.
with tcp_container.for_range(0, 16) as i:
    # code omitted for brevity
    . . .
# for_range usage for multi-core task launching.
with tcp_container.for_range(0, 2, task_num=16, task_type=TaskType.UNION2) as task_id:
    # code omitted for brevity
    . . .
```

7.5.2 if_scope

if_scope(

cond

)

创建一个 BANGPy 的 if 语句，当满足 cond 所描述的条件时，执行作用域中的语句。

参数说明

- cond: 判断条件是否成立的表达式。

返回值描述

- if 语句作用域。

注意事项

- if_scope 在 with 中使用, 请参见示例。
- 参数 cond 是 int、Scalar 类型的变量组成的条件表达式，请参见示例。
- 在 cond 中，请使用 all、any 表达逻辑与、逻辑或，参见示例。

示例

```
bp = tcp.TCP()
i = bp.Scalar(name='i', dtype=bangpy.int32, value=8)
```

```
x = bp.Tensor(shape=[10,], dtype=bangpy.float32, name="x", scope="nram")
with bp.if_scope((i % 2) == 0):
    x[i] = x[i - 1] + 1

j = bp.Scalar(name='j', dtype=bangpy.int32, value=8)
with bp.if_scope(bp.all(i == j, i > 1)):
    ...

with bp.if_scope(bp.any(i == j, i > 1)):
    ...
```

7.5.3 else_scope

else_scope()

创建一个 BANGPy 的 else 语句，与 if 语句配合使用。

返回值描述

- else_scope: else 作用域。

注意事项

- else_scope 必须与 if_scope 配合使用。

示例

```
tcp = tcp.TCP()
i = bp.Scalar(name='i', dtype=bangpy.int32, value=8)
x = tcp.Tensor(shape=[10,], dtype=bangpy.float32, name=x, scope="nram")
with tcp.if_scope((i % 2) == 0):
    x[i] = x[i - 1] + 1
with tcp.else_scope():
    . . .
```

7.6 编译接口

7.6.1 BuildBANG

BuildBANG(

inputs,

outputs=None,

dump_ir=False,

kernel_name=""

)

将 TCP 容器中的计算描述编译成可执行 module。

参数说明

- inputs: 输入张量，类型: list of Tensor。
- outputs: 输出张量，类型: list of Tensor。
- dump_ir: 编译选项，类型: bool。
- kernel_name: 生成的 Kernel 的名称，类型: str。

返回值描述

- 可执行 module，类型: BANGModule。

注意事项

- kernel_name 用户必须指定，不能为空字符串。

示例

```
fvec_add = bp.BuildBANG(inputs=[tensor_in0, tensor_in1],
                        outputs=[tensor_out],
                        dump_ir=True,
                        kernel_name="fvec_add")
```


7.7 运行模块接口

7.7.1 调用

BANGModule(

args,

...

)

启用 MLU 设备运行。

参数说明

- args: 输入与输出数据，个数不确定。类型: bangpy.Array。

返回值描述

- 无返回值。

注意事项

- 参数 args 的顺序必须与 BuildBANG 的中的 [inputs, outputs] 顺序对应。

示例

```
fvec_add = bp.BuildBANG(inputs=[tensor_in0, tensor_in1],
                        outputs=[tensor_out],
                        dump_ir=True,
                        kernel_name="fvec_add")
fvec_add(a, b, c)
```

7.8 调试接口

7.8.1 print

print(

input_data

)

打印输入数据，可用于 debug。

参数说明

- input_data: 需要被打印的数据。类型: Scalar, Tensor, itvar, str, int。

返回值描述

- 无返回值。

注意事项

- 无。

示例

```
Input1 = bp.Scalar(name='Input1', dtype=bangpy.float32, value=6)
                        kernel_name="fvec_add")
bp.print(Input1)
```

7.8.2 time_evaluator

time_evaluator(

ctx=tvm.mlu(0),

number=10,

repeat=1,

min_repeat_ms=0

)

评估程序运行时间。

参数说明

- ctx: 程序运行的 ctx，默认是 tvm.mlu(0)，类型: TVMContext。
- number: 程序运行次数，默认跑 10 次，类型: int。
- repeat: 重复测量的次数，类型: int。
- min_repeat_ms: 最小 repeat 周期，类型: int。

返回值描述

- 返回可执行函数。

注意事项

- 无。

示例

```
fvec_add = bp.BuildBANG(inputs=[tensor_in0, tensor_in1],
                        outputs=[tensor_out],
                        dump_ir=True,
                        kernel_name="fvec_add")
evaluator = fvec_add.time_evaluator(number=10, repeat=1, min_repeat_ms=0)
print('add : %f ms' % (evaluator(a, b, c).mean * 1e3))
```


8 TensorOp 接口

8.1 张量接口

8.1.1 创建张量

tensor(

shape,

dtype,

name

)

创建一个 TensorOp 张量。

参数说明

- shape: 张量形状。类型: tuple of int。
- dtype: 张量元素的数值类型。参见数值类型 节。
- name: 张量名字。类型: str。

返回值描述

- 创建的张量。类型: tensor_op.Tensor。

注意事项

- TensorOp 中的张量与 TCP 中的张量不可混用。
- 张量的名字用于后续运行时设置参数。详见运行模块接口 。

示例

```
from bangpy import tensor_op as tsop


input0 = tsop.tensor(shape=(n, h, w, c), dtype=bangpy.float16, name="input0")
```


8.2 单目运算算子

8.2.1 abs

abs(

src_data

)

对输入张量中每个元素取绝对值。

参数说明

- src_data: 源操作数，类型: tensor_op.Tensor。

返回值描述

- 计算结果。类型: tensor_op.Tensor。

注意事项

- 支持的数值类型: float16, float32。

示例

```
from bangpy import tesnor_op as tsop

input0 = tsop.tensor(shape, dtype, name)
output = tsop.abs(input0)
```

8.2.2 cos

cos(

src_data

)

对输入张量中每个元素计算余弦值。

参数说明

- src_data: 源操作数，类型: tensor_op.Tensor。

返回值描述

- 计算结果。类型: tensor_op.Tensor。

注意事项

- 支持的数值类型: float16, float32。
- 输入数据范围：$(-2 * \pi, 2 * \pi)$ 。

示例

```
from bangpy import tesnor_op as tsop

input0 = tsop.tensor(shape, dtype, name)
output = tsop.cos(input0)
```

8.2.3 exp

exp(

src_data

)

对输入张量中每个元素进行以 e 为底数的幂运算。

参数说明

- src_data: 源操作数，类型: tensor_op.Tensor。

返回值描述

- 计算结果。类型: tensor_op.Tensor。

注意事项

- 支持的数值类型: float16, float32。
- 输入数据范围：(-7.75, 10)。

示例

```
from bangpy import tesnor_op as tsop

input0 = tsop.tensor(shape, dtype, name)
output = tsop.exp(input0)
```

8.2.4 exp2

exp2(

src_data

)

对输入张量中每个元素进行以 2 为底数的幂运算。

参数说明

- src_data: 源操作数，类型: tensor_op.Tensor。

返回值描述

- 计算结果。类型: tensor_op.Tensor。

注意事项

- 支持的数值类型: float16, float32。
- 当数据类型是 float16 时，输入数据范围：(0, 16)。
- 当数据类型是 float32 时，输入数据范围：(0, 32)。

示例

```
from bangpy import tesnor_op as tsop

input0 = tsop.tensor(shape, dtype, name)
output = tsop.exp2(input0)
```

8.2.5 gelu

gelu(

src_data

)

对输入张量使用 gelu 激活函数激活。

参数说明

- src_data: 源操作数，类型: tensor_op.Tensor。

返回值描述

- 计算结果。类型: tensor_op.Tensor。

注意事项

- 支持的数值类型: float16, float32。

示例

```
from bangpy import tesnor_op as tsop

input0 = tsop.tensor(shape, dtype, name)
output = tsop.gelu(input0)
```

8.2.6 log

log(

src_data

)

对输入张量中每个元素取以 e 为底数的对数。

参数说明

- src_data: 源操作数，类型: tensor_op.Tensor。

返回值描述

- 计算结果。类型: tensor_op.Tensor。

注意事项

- 支持的数值类型: float16, float32。
- 输入数据范围：(1, 60000)。

示例

```
from bangpy import tesnor_op as tsop

input0 = tsop.tensor(shape, dtype, name)
output = tsop.log(input0)
```

8.2.7 logical_not

logical_not(

src_data

)

对输入张量中每个元素，为零则输出张量对应位置为 1，非零则输出张量对应位置为 0。

参数说明

- src_data: 源操作数，类型: tensor_op.Tensor。

返回值描述

- 计算结果。类型: tensor_op.Tensor。

注意事项

- 支持的数值类型: float16, float32。

示例

```
from bangpy import tesnor_op as tsop

input0 = tsop.tensor(shape, dtype, name)
output = tsop.logical_not(input0)
```

8.2.8 reciprocal

reciprocal(

src_data

)

对输入张量中每个元素取倒数。

参数说明

- src_data: 源操作数，类型: tensor_op.Tensor。

返回值描述

- 计算结果。类型: tensor_op.Tensor。

注意事项

- 支持的数值类型: float16, float32。
- 输入数据范围：(0.01, 500)。

示例

```
from bangpy import tesnor_op as tsop

input0 = tsop.tensor(shape, dtype, name)
output = tsop.reciprocal(input0)
```

8.2.9 relu

relu(

src_data

)

对输入张量使用 ReLU 激活函数激活。

参数说明

- src_data: 源操作数，类型: tensor_op.Tensor。

返回值描述

- 计算结果。类型: tensor_op.Tensor。

注意事项

- 支持的数值类型: float16, float32。

示例

```
from bangpy import tesnor_op as tsop

input0 = tsop.tensor(shape, dtype, name)
output = tsop.relu(input0)
```

8.2.10 rsqrt

rsqrt(

src_data

)

对输入张量中每个元素开方取倒数。

参数说明

- src_data: 源操作数，类型: tensor_op.Tensor。

返回值描述

- 计算结果。类型: tensor_op.Tensor。

注意事项

- 支持的数值类型: float16, float32。
- 输入数据范围：$(5 * 10^{-3}, 63487)$ 。

示例

```
from bangpy import tesnor_op as tsop

input0 = tsop.tensor(shape, dtype, name)
output = tsop.rsqrt(input0)
```

8.2.11 sigmoid

sigmoid(

src_data

)

对输入张量使用 sigmoid 激活函数激活。

参数说明

- src_data: 源操作数，类型: tensor_op.Tensor。

返回值描述

- 计算结果。类型: tensor_op.Tensor。

注意事项

- 支持的数值类型: float16, float32。

示例

```
from bangpy import tesnor_op as tsop

input0 = tsop.tensor(shape, dtype, name)
output = tsop.sigmoid(input0)
```

8.2.12 sign

sign(

src_data

)

对输入张量中每个元素，为零则输出张量对应位置为 0，为正数则输出张量对应位置为 1，为负数则输出张量对应位置为-1。

参数说明

- src_data: 源操作数，类型: tensor_op.Tensor。

返回值描述

- 计算结果。类型: tensor_op.Tensor。

注意事项

- 支持的数值类型: float16, float32。

示例

```
from bangpy import tesnor_op as tsop

input0 = tsop.tensor(shape, dtype, name)
output = tsop.sign(input0)
```

8.2.13 sin

sin(

src_data

)

对输入张量中每个元素取正弦值。

参数说明

- src_data: 源操作数，类型: tensor_op.Tensor。

返回值描述

- 计算结果。类型: tensor_op.Tensor。

注意事项

- 支持的数值类型: float16, float32。
- 输入数据范围：$(-2 * \pi, 2 * \pi)$。

示例

```
from bangpy import tesnor_op as tsop

input0 = tsop.tensor(shape, dtype, name)
output = tsop.sin(input0)
```

8.2.14 sqrt

sqrt(

src_data

)

对输入张量中每个元素开方。

参数说明

- src_data: 源操作数，类型: tensor_op.Tensor。

返回值描述

- 计算结果。类型: tensor_op.Tensor。

注意事项

- 支持的数值类型: float16, float32。
- 输入数据范围：(0, 65504)。

示例

```
from bangpy import tesnor_op as tsop

input0 = tsop.tensor(shape, dtype, name)
output = tsop.sqrt(input0)
```


8.2.15 square

square(

src_data

)

对输入张量中每个元素取平方。

参数说明

- src_data: 源操作数，类型: tensor_op.Tensor。

返回值描述

- 计算结果。类型: tensor_op.Tensor。

注意事项

- 支持的数值类型: float16, float32。

示例

```
from bangpy import tesnor_op as tsop

input0 = tsop.tensor(shape, dtype, name)
output = tsop.square(input0)
```

8.2.16 tanh

tanh(

src_data

)

对输入张量中每个元素取双曲正切值。

参数说明

- src_data: 源操作数，类型: tensor_op.Tensor。

返回值描述

- 计算结果。类型: tensor_op.Tensor。

注意事项

- 支持的数值类型: float16, float32。
- 输入数据范围：(0.01, 500)。

示例

```
from bangpy import tesnor_op as tsop

input0 = tsop.tensor(shape, dtype, name)
output = tsop.tanh(input0)
```

8.3 双目运算算子

8.3.1 add

add(

lhs,

rhs

)

两个张量元素对应相加，或者一个张量中每个元素加常量。

参数说明

- lhs: 源操作数，类型: tensor_op.Tensor。
- rhs: 源操作数，类型: tensor_op.Tensor 或常数（numbers.Number）。

返回值描述

- 计算结果。类型: tensor_op.Tensor。

注意事项

- 支持的数值类型: float16, float32。
- 当左右操作数都是张量时，其形状、数据类型必须完全相同。
- 当一个操作数为常数时，另一个操作数张量至少是二维。
- 用户可以使用运算符代替接口调用。

示例

```
from bangpy import tesnor_op as tsop

lhs = tsop.tensor(shape, dtype, name)
rhs = tsop.tensor(shape, dtype, name)
output = tsop.add(lhs, rhs)
output = tsop.add(lhs, 1)
output = lhs + rhs
output = 1 + rhs
output = lhs + 1
```

8.3.2 multiply

multiply(

lhs,

rhs

)

两个张量元素对应相乘，或者一个张量中每个元素乘常量。

参数说明

- lhs: 源操作数，类型: tensor_op.Tensor。
- rhs: 源操作数，类型: tensor_op.Tensor 或常数（numbers.Number）。

返回值描述

- 计算结果。类型: tensor_op.Tensor。

注意事项

- 支持的数值类型: float16, float32。
- 当左右操作数都是张量时，其形状、数据类型必须完全相同。
- 当一个操作数为常数时，另一个操作数张量至少是二维。
- 用户可以使用运算符代替接口调用。

示例

```
from bangpy import tesnor_op as tsop

lhs = tsop.tensor(shape, dtype, name)
rhs = tsop.tensor(shape, dtype, name)
output = tsop.multiply(lhs, rhs)
output = tsop.multiply(lhs, 1)
output = lhs * rhs
output = lhs * 1
output = 1 * rhs
```

8.3.3 subtract

subtract(
lhs,
rhs
)

两个张量元素对应相减，或者一个张量中每个元素减常量。

参数说明

- lhs: 源操作数，类型: tensor_op.Tensor。
- rhs: 源操作数，类型: tensor_op.Tensor 或常数（numbers.Number）。

返回值描述

- 计算结果。类型: tensor_op.Tensor。

注意事项

- 支持的数值类型: float16, float32。
- 当左右操作数都是张量时，其形状、数据类型必须完全相同。
- 当一个操作数为常数时，另一个操作数张量至少是二维。
- 用户可以使用运算符代替接口调用。

示例

```
from bangpy import tesnor_op as tsop

lhs = tsop.tensor(shape, dtype, name)
rhs = tsop.tensor(shape, dtype, name)
output = tsop.subtract(lhs, rhs)
output = tsop.subtract(lhs, 1)
output = lhs - rhhs
output = lhs - 1
output = 1 - rhs
```

8.3.4 divide

divide(

lhs,

rhs

)

两个张量元素对应相除。

参数说明

- lhs: 源操作数，类型: tensor_op.Tensor。
- rhs: 源操作数，类型: tensor_op.Tensor。

返回值描述

- 计算结果。类型: tensor_op.Tensor。

注意事项

- 支持的数值类型: float16, float32。
- 左右操作数形状、数据类型必须完全相同。
- 当一个操作数为常数时，另一个操作数张量至少是二维。
- 除数数据范围为 (0.01, 500)。
- 用户可以使用运算符代替接口调用。

示例

```
from bangpy import tesnor_op as tsop

lhs = tsop.tensor(shape, dtype, name)
rhs = tsop.tensor(shape, dtype, name)
```

```
output = tsop.divide(lhs, rhs)
output = lhs / rhs
output = 2 / rhs
output = lhs / 2
```

8.3.5 logical_and

logical_and(

lhs,

rhs

)

两个张量元素对应进行逻辑与操作。

参数说明

- lhs: 源操作数，类型: tensor_op.Tensor。
- rhs: 源操作数，类型: tensor_op.Tensor。

返回值描述

- 计算结果。类型: tensor_op.Tensor。

注意事项

- 支持的数值类型: float16, float32。
- 左右操作数形状、数据类型必须完全相同。

示例

```
from bangpy import tesnor_op as tsop

lhs = tsop.tensor(shape, dtype, name)
rhs = tsop.tensor(shape, dtype, name)
output = tsop.logical_and(lhs, rhs)
```

8.3.6 logical_or

logical_or(

lhs,

rhs

)

两个张量元素对应进行逻辑或操作。

参数说明

- lhs: 源操作数，类型: tensor_op.Tensor。
- rhs: 源操作数，类型: tensor_op.Tensor。

返回值描述

- 计算结果。类型: tensor_op.Tensor。

注意事项

- 支持的数值类型: float16, float32。
- 左右操作数形状、数据类型必须完全相同。

示例

```
from bangpy import tesnor_op as tsop

lhs = tsop.tensor(shape, dtype, name)
rhs = tsop.tensor(shape, dtype, name)
output = tsop.logical_or(lhs, rhs)
```

8.3.7 equal

equal(

lhs,

rhs

)

两个张量元素对应判断是否相等，相等则输出张量对应位置为 1，否则为 0。

参数说明

- lhs: 源操作数，类型: tensor_op.Tensor。
- rhs: 源操作数，类型: tensor_op.Tensor。

返回值描述

- 计算结果。类型: tensor_op.Tensor。

注意事项

- 支持的数值类型: float16, float32。
- 左右操作数形状、数据类型必须完全相同。

示例

```
from bangpy import tesnor_op as tsop

lhs = tsop.tensor(shape, dtype, name)
rhs = tsop.tensor(shape, dtype, name)
output = tsop.equal(lhs, rhs)
```

8.3.8 not_equal

not_equal(

lhs,

rhs

)

两个张量元素对应判断是否相等，相等则输出张量对应位置为 0，否则为 1。

参数说明

- lhs: 源操作数，类型: tensor_op.Tensor。
- rhs: 源操作数，类型: tensor_op.Tensor。

返回值描述

- 计算结果。类型: tensor_op.Tensor。

注意事项

- 支持的数值类型: float16, float32。
- 左右操作数形状、数据类型必须完全相同。

示例

```
from bangpy import tesnor_op as tsop

lhs = tsop.tensor(shape, dtype, name)
rhs = tsop.tensor(shape, dtype, name)
output = tsop.not_equal(lhs, rhs)
```

8.3.9 greater

greater(

lhs,

rhs

)

两个张量元素对应比较大小，左操作数中元素大于右操作数中元素则输出张量对应位置为 1，否则为 0。

参数说明

- lhs: 源操作数，类型: tensor_op.Tensor。
- rhs: 源操作数，类型: tensor_op.Tensor。

返回值描述

- 计算结果。类型: tensor_op.Tensor。

注意事项

- 支持的数值类型: float16, float32。
- 左右操作数形状、数据类型必须完全相同。

示例

```
from bangpy import tesnor_op as tsop

lhs = tsop.tensor(shape, dtype, name)
rhs = tsop.tensor(shape, dtype, name)
output = tsop.greater(lhs, rhs)
```

8.3.10 greater_equal

greater_equal(

lhs,

rhs

)

两个张量元素对应比较大小，左操作数中元素大于等于右操作数中元素则输出张量对应位置为 1，否则为 0。

参数说明

- lhs: 源操作数，类型: tensor_op.Tensor。
- rhs: 源操作数，类型: tensor_op.Tensor。

返回值描述

- 计算结果。类型: tensor_op.Tensor。

注意事项

- 支持的数值类型: float16, float32。
- 左右操作数形状、数据类型必须完全相同。

示例

```
from bangpy import tesnor_op as tsop

lhs = tsop.tensor(shape, dtype, name)
rhs = tsop.tensor(shape, dtype, name)
output = tsop.greater_equal(lhs, rhs)
```

8.3.11 less

less(

lhs,

rhs

)

两个张量元素对应比较大小，左操作数中元素小于右操作数中元素则输出张量对应位置为 1，否则为 0。

参数说明

- lhs: 源操作数，类型: tensor_op.Tensor。
- rhs: 源操作数，类型: tensor_op.Tensor。

返回值描述

- 计算结果。类型: tensor_op.Tensor。

注意事项

- 支持的数值类型: float16, float32。
- 左右操作数形状、数据类型必须完全相同。

示例

```
from bangpy import tesnor_op as tsop

lhs = tsop.tensor(shape, dtype, name)
rhs = tsop.tensor(shape, dtype, name)
output = tsop.less(lhs, rhs)
```

8.3.12 less_equal

less_equal(

lhs,

rhs

)

两个张量元素对应比较大小，左操作数中元素小于等于右操作数中元素则输出张量对应位置为 1，否则为 0。

参数说明

- lhs: 源操作数，类型: tensor_op.Tensor。
- rhs: 源操作数，类型: tensor_op.Tensor。

返回值描述

- 计算结果。类型: tensor_op.Tensor。

注意事项

- 支持的数值类型: float16, float32。
- 左右操作数形状、数据类型必须完全相同。

示例

```
from bangpy import tesnor_op as tsop

lhs = tsop.tensor(shape, dtype, name)
rhs = tsop.tensor(shape, dtype, name)
output = tsop.less_equal(lhs, rhs)
```

8.3.13 maximum

maximum(

lhs,

rhs

)

选出两个输入张量对应位置两个元素中的较大值，作为输出张量对应位置的值。

参数说明

- lhs: 源操作数，类型: tensor_op.Tensor。
- rhs: 源操作数，类型: tensor_op.Tensor。

返回值描述

- 计算结果。类型: tensor_op.Tensor。

注意事项

- 支持的数值类型: float16, float32。
- 左右操作数形状、数据类型必须完全相同。

示例

```
from bangpy import tesnor_op as tsop

lhs = tsop.tensor(shape, dtype, name)
rhs = tsop.tensor(shape, dtype, name)
output = tsop.maximum(lhs, rhs)
```

8.3.14 minimum

minimum(

lhs,

rhs

)

选出两个输入张量对应位置两个元素中的较小值，作为输出张量对应位置的值。

参数说明

- lhs: 源操作数，类型: tensor_op.Tensor。
- rhs: 源操作数，类型: tensor_op.Tensor。

返回值描述

- 计算结果。类型: tensor_op.Tensor。

注意事项

- 支持的数值类型: float16, float32。
- 左右操作数形状、数据类型必须完全相同。

示例

```
from bangpy import tesnor_op as tsop

lhs = tsop.tensor(shape, dtype, name)
rhs = tsop.tensor(shape, dtype, name)
output = tsop.minimum(lhs, rhs)
```

8.4 编译接口

8.4.1 BuildBANG

BuildBANG(

inputs,

outputs,

target,

dump_ir=False,

)

将 TensorOp 的计算描述编译成运行时 module。

参数说明

- inputs: 输入张量列表，类型: list of tensor_op.Tensor。
- outputs: 输出张量列表，类型: list of tensor_op.Tensor。
- target: 硬件架构名称，例如 MLU270, MLU290, MLU370 ，类型：str。
- dump_ir: 编译选项，表示是否保存中间表示，类型: bool。

返回值描述

- 运行时 module，类型: BANGGraphModule。

示例

```
from bangpy import tensor_op as tsop
fvec_add = tsop.BuildBANG(inputs=[tensor_in0, tensor_in1],
                          outputs=[tensor_out],
                          target="MLU270",
                          dump_ir=False)
```

8.5 运行模块接口

8.5.1 输入数据

BANGGraphModule.set_input(

****param**

)

设置输入数据。

参数说明

- param: 输入参数列表。类型：dict。

返回值描述

- 无。

注意事项

- 参数写成 param_name0=data0, param_name1=data1, … 的形式，其中 param_name0, param_name1 是 TensorOp 描述计算时声明的张量的名字，data0，data1 是 bangpy.Array 或 numpy.ndarray 类型的输入数据。

示例

```
from bangpy import tensor_op as tsop
fvec_add = tsop.BuildBANG(inputs=[tensor_in0, tensor_in1],
                          outputs=[tensor_out],
                          target="MLU270",
                          dump_ir=False)
fvec_add.set_input(param_name0=np.random.uniform(...),
                   param_name1=param1,
                   param_name2=bangpy.Array(param2, bangpy.context(0)), ...)
```

8.5.2 运行

BANGGraphModule.run(

)

启用 MLU 设备运行。

参数说明

- 无。

返回值描述

- 无。

示例

```
fvec_add.run()
```

8.5.3 获得输出数据

BANGGraphModule.get_output(

index

)

获得输出数据。

参数说明

- index: 输出的索引。类型: int。

返回值描述

- output: 输出数据，调用 asnumpy() 以获得 numpy.ndarray 类型数据。

示例

```
mlu_output = fvec_add.get_output(0)
```

9 通用接口

9.1 运行模块接口

9.1.1 save

save(

dirname

)

将可执行 Module 在给定目录中保存为 so 文件和 mlu 文件。

参数说明

- dirname: 用来保存 so 文件和 mlu 文件的目录的路径。类型: str。

返回值描述

- 无返回值。

注意事项

- 用户需保证 dirname 所指定的目录为空。

示例

```
fvec_add = bp.BuildBANG(inputs=[tensor_in0, tensor_in1],
                        outputs=[tensor_out],
                        dump_ir=True,
                        kernel_name="fvec_add")
fvec_add.save(dirname)

from bangpy import tensor_op as tsop
fvec_add = tsop.BuildBANG(inputs=[tensor_in0, tensor_in1],
                          outputs=[tensor_out],
                          dump_ir=True,
                          target="MLU270")
fvec_add.save(dirname)
```

9.1.2 load_module

load_module(

dirname,

target

)

从给定目录中载入 so 文件和 mlu 文件，根据指定的架构 target，形成可执行 module。

参数说明

- dirname: 用来保存 so 文件和 mlu 文件的目录的路径。类型: str。
- target: 用来指定形成的可执行 module 基于的架构。

返回值描述

- 可执行 module。类型: BANGModule 或 BANGGraphModule。

注意事项

- 用户只能读取 dirname 中的文件，不可随意修改其中文件名和内容，以免影响 load_module 的正常功能。

示例

```
from bangpy import load_module
floaded = load_module(dirname, target)
# TCP
floaded(a, b, c)
# TensorOp
floaded.set_input(param_name0=a, param_name1=b)
floaded.run()
c = floaded.get_output(0)
```

9.2 芯片信息接口

9.2.1 get_ram_size

get_ram_size(

ram_scope

)

获取片上存储可用空间大小。

参数说明

- ram_scope: 片上存储类型。类型: str。

返回值描述

- 空间大小，以字节为单位。类型: int。

注意事项

- 目前 ram_scope 仅允许为 `nram` , `wram` 。

示例

```
from bangpy.platform.bang_config import MLUConfig
nram_size = MLUConfig().get_ram_size("nram")
wram_size = MLUConfig().get_ram_size("wram")
```