
TPU-KERNEL 开发参考手册

SOPHGO

2024 年 05 月 10 日

1	简介	2
1.1	介绍	2
2	TPU 架构	3
2.1	TPU 架构图	3
2.2	TPU 的内存类型	4
3	TPU 编程模型	5
3.1	TPU 的工作模式	5
3.2	TPU 编程模型	5
3.3	Host 端	7
3.4	Device 端	10
4	TPU API	23
4.1	基础定义	23
4.2	舍入模式	26
4.3	功能函数	28
4.4	数据同步函数	32
4.5	辅助函数	33
4.6	GDMA 操作	35
4.7	基础数据操作	62
4.8	数据类型转换与舍入操作	65
4.9	一元操作	68
4.10	二元操作	78
4.11	浮点二元操作	102
4.12	整型二元操作	116
4.13	比较选择函数	134
4.14	浮点矩阵操作	137
4.15	整型矩阵操作	146
4.16	浮点神经网络操作	166
4.17	定点神经网络操作	177
4.18	激活函数	179
4.19	scatter 和 gather 操作	188
4.20	特殊函数	198
4.21	量化操作	203
4.22	HAU 操作	208



法律声明

- 版权所有 © 算能 2024. 保留一切权利。
- 非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

注意

- 您购买的产品、服务或特性等应受算能商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，算能对本文档内容不做任何明示或默示的声明或保证。由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

技术支持

- **地址：**北京市海淀区丰豪东路 9 号院中关村集成电路设计园（ICPARK）1 号楼
- **邮编：**100094
- **网址：**<https://www.sophgo.com/>
- **邮箱：**sales@sophgo.com
- **电话：**+86-10-57590723 +86-10-57590724

1.1 介绍

TPUKernel 是算丰 BM1684x 设备底层开发接口，利用该接口可以完成如下操作：

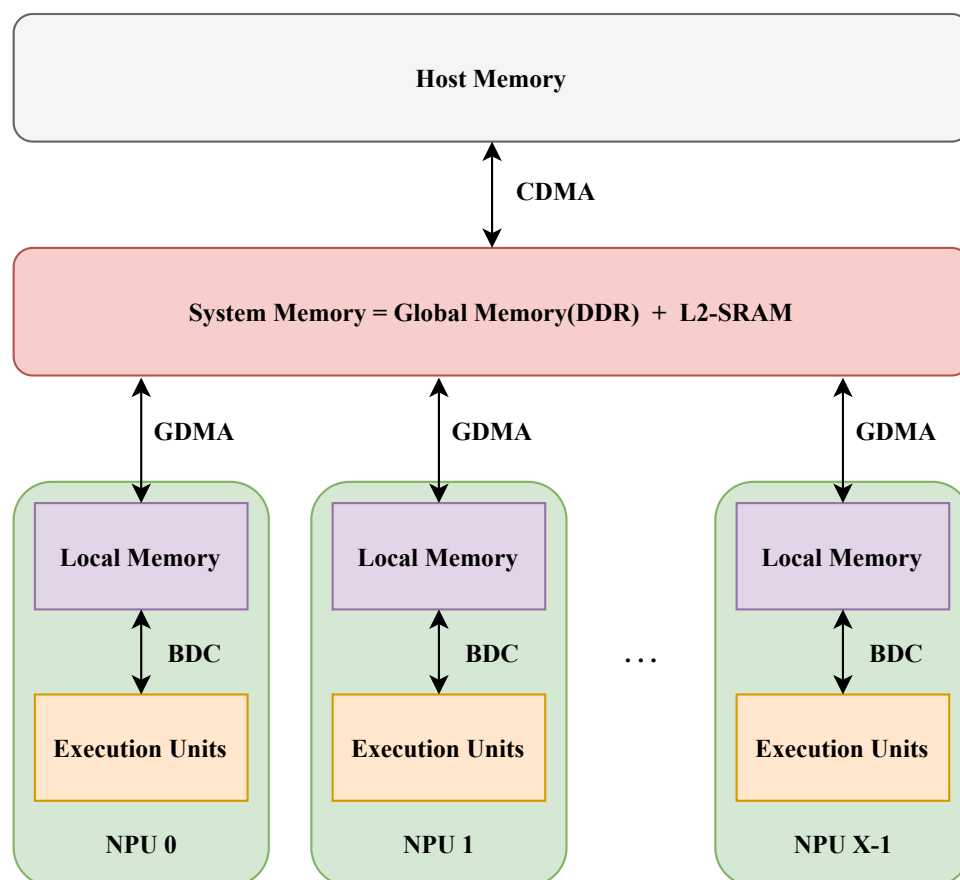
- 1. 调用专用指令（如 convolution、pooling 等），实现深度学习业务逻辑的加速
 - 2. 调用通用指令（如矩阵乘法等），实现用户定制的各种算法加速。
-

该文档包含以下几方面内容：

- 1. BM1684x 设备架构介绍。
 - 2. BM1684x 设备编程接口 API（指令集封装）。
-

2.1 TPU 架构图

下面是 BM1684x 设备的 TPU 架构图.



如上图所示, TPU 是一种多个计算核的架构设计, 每一个核被称之为 **NPU (Neural network Processing Unit)**。

TPU 按照单指令多数据 (Single Instruction Multiple Data, SIMD) 的模式进行计算, 即在某一时刻所有的 NPU 都会执行同样的计算指令, 但是每一个 NPU 操作的数据不一样。

在每一个 NPU 内部存储数据的内存被称之为 Local Memory, 每个 NPU 的计算单元只能访问 Local Memory。

TPU 进行计算加速通常分为以下几步:

1. 将数据从主机端内存搬运到 TPU 的系统内存 (Global Memory) 当中,
 2. 再将数据从系统内存 (Global memory) 再搬运到 Local Memory 当中,
 3. 驱动计算单元对 Local Memory 当中的数据进行计算, 并将计算结果返回 Local Memory,
 4. 将计算结果从 Local Memory 搬运回 Global Memory,
 5. 将 Global Memory 中的计算结果搬运回主机端内存。
-

2.2 TPU 的内存类型

在上面的介绍中, 该 TPU 架构下, 数据主要存在于以下的内存中:

- **系统内存 (System Memory)**
 - Global Memory: TPU 设备外的内存, DDR。
 - L2-SRAM: 片上内存, 作为缓存。
- **Local Memory**: 片上内存, BDC 计算单元直接访问的内存类型。

3.1 TPU 的工作模式

在算能的产品当中，根据主控单元（主机端 Host）的不同，TPU 对应有两种不同的运行模式，分别被称为 PCIe 模式和 SOC 模式。

- **PCIe 模式:** 该模式下对应的产品形态为 SC 系列板卡。板卡通过 PCIe 接口连接到主机服务器上，主机服务器作为主控单元（Host）控制板卡的运行。
 - **SoC 模式:** 该模式下对应的产品形态为 SE 系列边缘推理设备。推理设备上的包含一个 8 核的 A53 处理器作为设备的主控单元（Host）控制板卡的运行。
-

3.2 TPU 编程模型

由于 TPU 是一个异构的架构设计，由 **主机端** 发送指令，**设备端** 接收指令，按照指令执行指定的操作。因此，驱动 TPU 的进行指定计算需要分别完成主机端和设备端的两部分代码：

- **主机端 (Host):** 主机端代码，运行在主机侧，发送控制 TPU 运行的命令。
- **设备端 (Device):** 设备端代码，运行在设备侧，通常调用 TPU 的各种指令运行相应的运算。

主机端和设备端的代码由于目标设备不同，因此需要使用不同的编译器对代码进行编译。

PCIe 模式下:

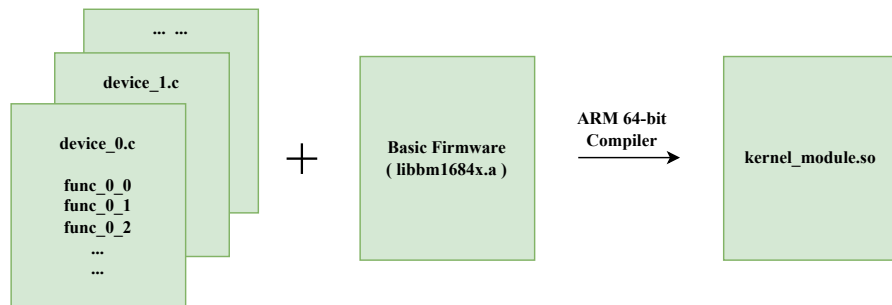
- 主机端代码编译器: 主机的本地 C++ 编译器。
- 设备端代码编译器: 用于 Linux ARM A53 的交叉编译器。

SoC 模式下, 代码在 x86 平台进行编译，在 ARM A53 平台运行。

- 主机端代码编译器: 用于 Linux ARM A53 交叉编译器。
- 设备端代码编译器: 用于 Linux ARM A53 交叉编译器。

上述 Linux ARM A53 交叉编译器，可以通过 TPUKernel 工具包内 scripts/prepare_toolchain.sh 脚本来进行下载

对于 Device 端代码，编译的过程可以被看作是动态链接库更新的过程，



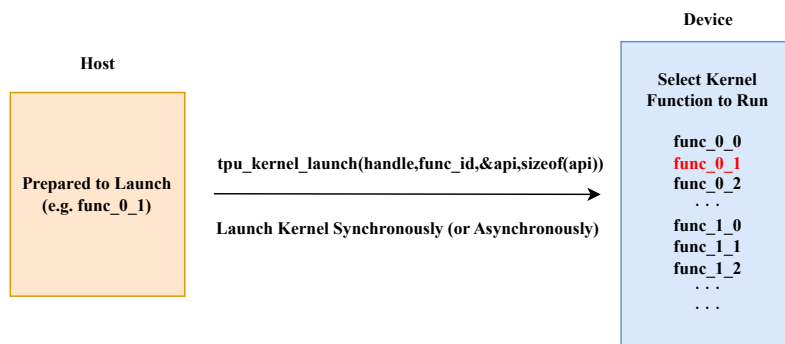
如上图所示，`device_*.c()` 就是开发者完成的设备端代码，设备端代码编译器将设备端代码和原始的底库 `libbm1684x.a` 进行链接，形成完整的 A53Lite 可加载的原始动态库文件。

3.3 Host 端

在完成了上述动态库的更新后，此时只需要加载动态库，然后在 Host 发送调用指令就可以控制 TPU 运行指定的计算。

```
typedef struct _param_func{
    ...
    ...
}__attribute__((packed)) param_func_0_1;

bm_handle_t handle;
param_func_0_1 param;
bm_dev_request(&handle, 0);
tpu_kernel_launch_sync(handle, "func_0_1", &param, sizeof(param));
bm_dev_free(&handle);
```



如上图所示，Device 端固件中已经注册了 `func_0_0()`，`func_0_1()`，`func_0_2()`，`func_1_0()`，`func_1_1()`，`func_1_2()` …等函数，主机端现在发送 `func_0_1()`，就可以调动 TPU 进行 `func_0_1()` 的计算。

主机端调用 TPU 进行计算有 **异步**和 **同步**两种方式。

3.3.1 同步运行方式

同步运行方式是指主机端在发送了调用计算命令后，不再继续下面代码的执行，直到 TPU 计算完成，继续进行主机端代码的执行。

主机端调用下面 API 后，便会等待 TPU 直到计算完成。

```
bm_status_t tpu_kernel_launch_sync(bm_handle_t handle, const char *func_name,
                                   const void *args, unsigned int size)
```

Launch the kernel function on device synchronously.

参数

- **handle** –Handle of the device.
- **func_name** –Name of the kernel function to launch on device.
- **args** –Pointer to the user-discript data package.
- **size** –Size of the user-discript data package in bytes.

返回 Status of launching kernel function, BM_SUCCESS means succeeded, otherwise, some errors caught.

3.3.2 异步运行方式

异步运行方式是指主机端在发送了调用指令后，继续执行后续的代码，TPU 和主机端异步运行。

主机端在调用了下面 API 后，TPU 就会开始计算，同时主机端仍会继续往下执行。

```
bm_status_t tpu_kernel_launch_async(bm_handle_t handle, const char *func_name,
                                   const void *args, unsigned int size)
```

Launch the kernel function on device asynchronously.

参数

- **handle** –Handle of the device.
- **func_name** –Name of the kernel function to launch on device.
- **args** –Pointer to the user-discript data package.
- **size** –Size of the user-discript data package in bytes.

返回 Status of launching kernel function, BM_SUCCESS means succeeded, otherwise, some errors caught.

主机端可以通过下面的 API 来实现和 TPU 的同步，调用下面 API 后，主机端会等待 TPU 完成计算。

```
bm_status_t tpu_kernel_sync(bm_handle_t handle)
```

Synchronize the device.

参数 **handle** –Handle of the device.

返回 Status of launching kernel function, BM_SUCCESS means succeeded, otherwise, some errors caught.

3.4 Device 端

上一节中，介绍了用户是怎样驱使 TPU 进行计算的。在这一章节，我们将开始学习如何编写 device 端代码，利用 TPU 完成我们想要的计算，在此之前我们需要首先了解一下，TPU 定义了哪些基本的指令。在 TPU 架构一节，我们已经介绍了 TPU 的整个计算过程，回顾一下，计算过程可以划分为三部分：

1. 数据在 Host 端内存和系统内存（System Memory）之间的来回搬运。
2. 数据在系统内存（System Memory）和 Local Memory 之间的来回搬运。
3. TPU 对 Local Memory 中的数据进行相关计算。

“数据在 Host 端内存和系统内存之间的来回搬运”，是主机端相关，这里不再涉及。由于数据存在于 Host 内存当中，因此这部分 API 通常由 Host 端来调用，关于 Host 端的相关命令可以参考上一小节。

3.4.1 指令系统

· GDMA 指令

系统内存和 Local-Memory 中与数据搬运相关的操作都由 GDMA 指令完成。与 GDMA 相关的指令都以 `tpu_gdma_()` 开头，包括数据在 Local Memory 之间的搬运、系统内存之间的搬运。详细的指令说明和参数，参见“TPU_API”章节。

· BDC 指令

TPU 进行数据计算的相关操作都可以由 BDC 指令来完成。与 BDC 相关的指令都以 `tpu_bdc_()` 开头。

· HAU 指令

一些不适用于并行加速计算的指令集，包括 NMS、SORT 等。

3.4.2 内存与数据排列

· 数据的表示

Tensor

在 TPU 中，我们使用 Tensor 来描述数据。

Tensor 是一个 4 维数组，使用 4 元组 (N,C,H,W) 来描述一个 Tensor 的几何尺寸 (**Shape**)。Tensor(n, c, h, w) 表示在 (n,c,h,w) 索引下的数据元素。

Stride 用于描述 Tensor 在实际内存当中是如何摆放，同样使用 4 元组 (N_stride, C_stride, H_stride, W_stride) 来描述，表示 **Tensor** 在内存当中存放时，元素间间隔了多少元素，具体而言：

- W_stride 描述的是从 Tensor (n,c,h,w) 到 Tensor (n,c,h,w+1) 两个元素之间，在内存存储时，间隔了多少个元素。
- H_stride 描述的是从 Tensor (n,c,h,w) 到 Tensor (n,c,h+1,w) 两个元素之间，在内存存储时，间隔了多少个元素。
- C_stride 描述的是从 Tensor (n,c,h,w) 到 Tensor (n,c+X,h,w) 两个元素之间，在内存存储时，间隔了多少个元素，X 表示 NPU 的数量。
- N_stride 描述的是从 Tensor (n,c,h,w) 到 Tensor (n+1,c,h,w) 两个元素之间，在内存存储时，间隔了多少个元素。

假设现在有一个 Tensor，它的每一个元素都占 1 个 byte，它的 Shape 是 (4, 3, 2, 2)，如果它的存储方式 Stride 是 (12, 4, 2, 1)，在内存当中，它的排列方式就会如下图所示，

Addr 0	ele	ele	ele	ele	ele	ele	ele	ele	ele	ele	ele	ele	Addr 11
Addr 12	ele	ele	ele	ele	ele	ele	ele	ele	ele	ele	ele	ele	Addr 23
Addr 24	ele	ele	ele	ele	ele	ele	ele	ele	ele	ele	ele	ele	Addr 35
Addr 36	ele	ele	ele	ele	ele	ele	ele	ele	ele	ele	ele	ele	Addr 47

如果 Tensor 的存储方式 Stride 是 (24, 4, 2, 1)，在内存当中，Tensor 的排列方式就会如下所示，

Addr 0	ele	ele	ele	ele	ele	ele	ele	ele	ele	ele	ele	ele	Addr 11
Addr 12													Addr 23
Addr 24	ele	ele	ele	ele	ele	ele	ele	ele	ele	ele	ele	ele	Addr 35
Addr 36													Addr 47
Addr 48	ele	ele	ele	ele	ele	ele	ele	ele	ele	ele	ele	ele	Addr 59
Addr 60													Addr 71
Addr 72	ele	ele	ele	ele	ele	ele	ele	ele	ele	ele	ele	ele	Addr 83
Addr 84													Addr 95

如果 Tensor 的存储方式 Stride 是 (24, 8, 4, 2)，在内存当中，Tensor 的排列方式就会如下所示，

Addr 0	ele		ele		ele		ele		ele		ele		Addr 11
Addr 12	ele		ele		ele		ele		ele		ele		Addr 23
Addr 24	ele		ele		ele		ele		ele		ele		Addr 35
Addr 36	ele		ele		ele		ele		ele		ele		Addr 47
Addr 48	ele		ele		ele		ele		ele		ele		Addr 59
Addr 60	ele		ele		ele		ele		ele		ele		Addr 71
Addr 72	ele		ele		ele		ele		ele		ele		Addr 83
Addr 84	ele		ele		ele		ele		ele		ele		Addr 95

如果它的存储方式 Stride 是 (24,8,4,1)，在内存当中，Tensor 的排列方式就会如下所示，

Addr 0	ele	ele			ele	ele			ele	ele			Addr 11
Addr 12	ele	ele			ele	ele			ele	ele			Addr 23
Addr 24	ele	ele			ele	ele			ele	ele			Addr 35
Addr 36	ele	ele			ele	ele			ele	ele			Addr 47
Addr 48	ele	ele			ele	ele			ele	ele			Addr 59
Addr 60	ele	ele			ele	ele			ele	ele			Addr 71
Addr 72	ele	ele			ele	ele			ele	ele			Addr 83
Addr 84	ele	ele			ele	ele			ele	ele			Addr 95

数据元素的类型

Stride 以元素个数作为计量单位。不同类型的数据元素占据不同的字节数，在 BM1684x 设备上支持如下格式的数据类型：

数据类型	字节数
INT8	1 Bytes
INT16	2 Bytes
INT32	4 Bytes
FP16	2 Bytes
BFP16	2 Bytes
FP32	4 Bytes

· Tensor 在 global memory 的排列方式

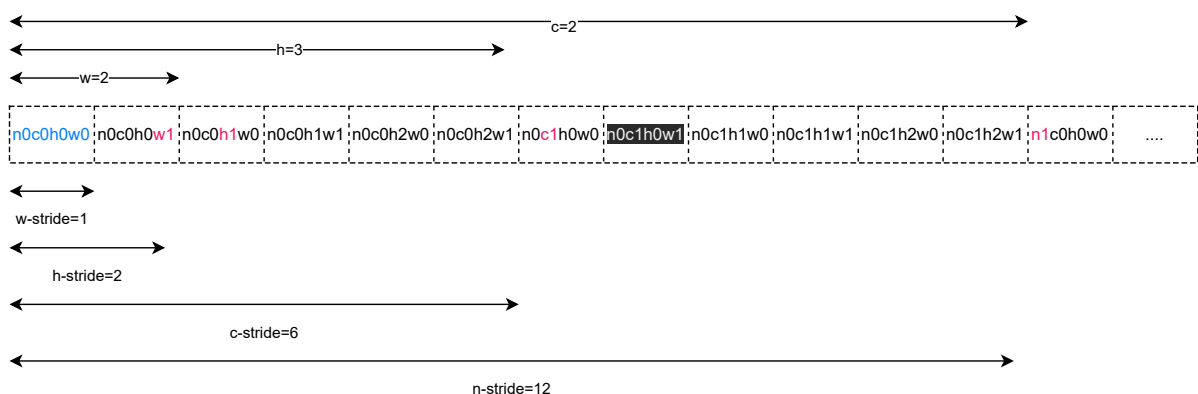
global memory 由一块 DDR 内存组成。

一个 Shape 为 (N, C, H, W) 的 Tensor，在 **global memory** 排列，对应的 Stride 为：

- $W_Stride = 1$,
- $H_Stride = W$,
- $C_Stride = H*W$,
- $N_Stride = C*H*W$ 。

这种排列方式被称为 **连续存储方式**。

举例：一个 Shape (N=2, C=2, H=3, W=2) 的 Tensor 在 global memory 排列方式



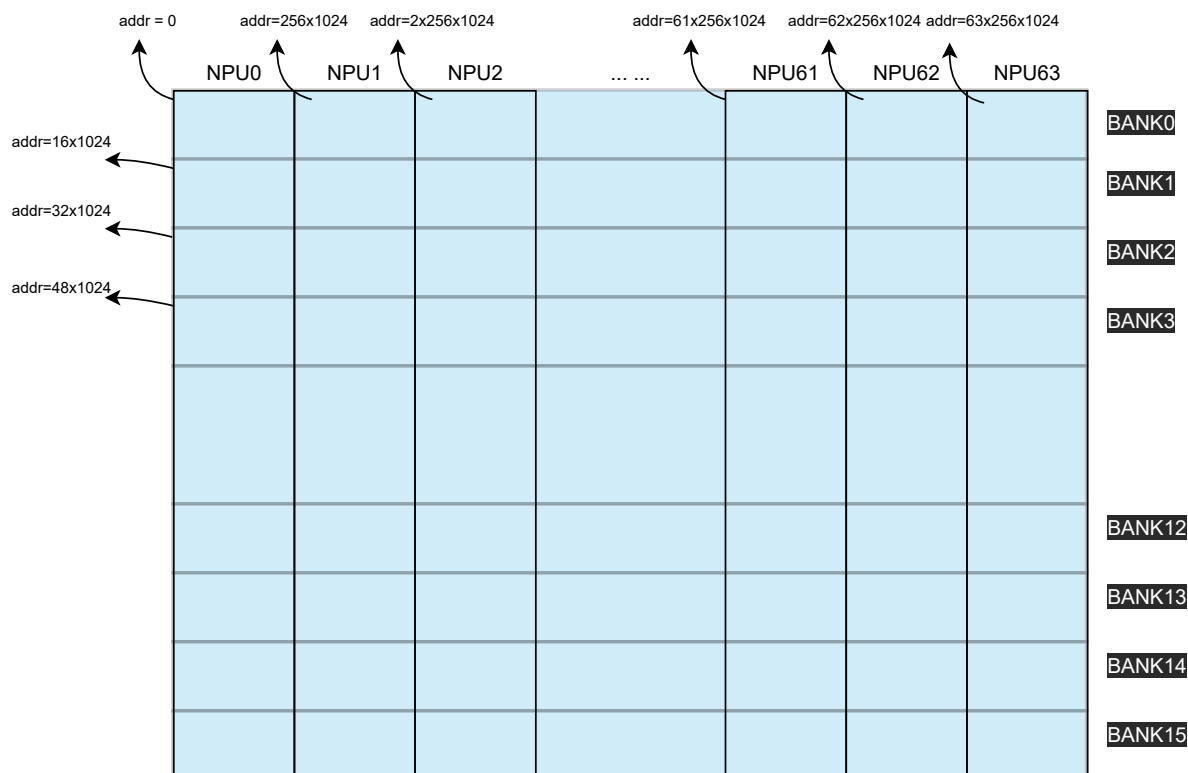
上图中，n0c0h0w0 代表 Tensor (0,0,0,0) 位置的元素。

· Tensor 在 local memory 的排列方式

local memory 的物理组成和地址分配

Local Memory 共由多片 SRAM（静态随机存取存储器）构成，每一片 SRAM 都被称为一个 **Bank**。bm1684x 设备一共由 16 个 Bank 构成。16 个 Bank 组成整个 **local memory**。

整个 Local Memory 同时被划分为了 64 个 **lane** (对应 64 个 NPU，以下用 NPU 指代 lane)，地址分配如下图所示：



BM1684x 的 Local Memory 大小为 **256KB * 64**，地址按照 NPU 进行分配，其中，NPU0 对应 **0 ~ 256*1024-1** 的地址，NPU1 对应 **256*1024 ~ 2*256*1024-1** 的地址，依次类推。

Tensor 在 Local memory 上排列的基本规则

Tensor 在 Local Memory 上的排布方式与 global Memory 的排布方式不同，主要区别在于 **C 维度的数据排布方式**。一个 Shape 为 (N, C, H, W) 的 Tensor，Tensor (N,c,H,W) 代表：当 $C = c$ 时，Tensor 的数据切片。对于不同的 c ，Tensor (N, c, H, W) 分配在不同的 NPU 上。

举例，Tensor 的 Shape (N=2,C=3,H=2,W=3), Stride (N_stride = 9, C_stride = 9, H_stride = 3, W_stride = 1) 那么 Tensor 在 Local Memory 上的数据排列方式如下所示。

NPU 0			NPU 1			NPU 2		
n0c0h0w0	n0c0h0w1	n0c0h0w2	n0c1h0w0	n0c1h0w1	n0c1h0w2	n0c2h0w0	n0c2h0w1	n0c2h0w2
n0c0h1w0	n0c0h1w1	n0c0h1w2	n0c1h1w0	n0c1h1w1	n0c1h1w2	n0c2h1w0	n0c2h1w1	n0c2h1w2
n1c0h0w0	n1c0h0w1	n1c0h0w2	n1c1h0w0	n1c1h0w1	n1c1h0w2	n1c2h0w0	n1c2h0w1	n1c2h0w2
n1c0h1w0	n1c0h1w1	n1c0h1w2	n1c1h1w0	n1c1h1w1	n1c1h1w2	n1c2h1w0	n1c2h1w1	n1c2h1w2

不同大小的 C 影响着实际存储方式，假设现在一共由 X 个 NPU，考虑下面几种存储情形：

情形 1: 当 Tensor 的 Shape 的维度 $C = X-1$ 时，当 Tensor 从 NPU0 开始存储时，那么 Tensor 在 Local Memory 上的排布方式如下所示。

NPU 0	NPU 1	NPU 2	...	NPU X - 2	NPU X - 1
(0, 0)	(0, 1)	(0, 2)	...	(0, X - 2)	
(1, 0)	(1, 1)	(1, 2)	...	(1, X - 2)	
⋮	⋮	⋮	⋮	⋮	
(N - 1, 0)	(N - 1, 1)	(N - 1, 2)	...	(N-1,X-2)	

在 NPU X-1 的地址空间，没有数据分布。

情形 2: 当 Tensor 的 Shape 的维度 $C = X-1$ 时，当 Tensor 从 NPU1 开始存储时，那么 Tensor 在 Local Memory 上的排布方式如下所示。

NPU 0	NPU 1	NPU 2	...	NPU X - 2	NPU X - 1
	(0, 0)	(0, 1)	(0, 2)	...	(0, X - 2)
	(1, 0)	(1, 1)	(1, 2)	...	(1, X - 2)
	⋮	⋮	⋮	⋮	⋮
	(N - 1, 0)	(N - 1, 1)	(N - 1, 2)	...	(N-1,X-2)

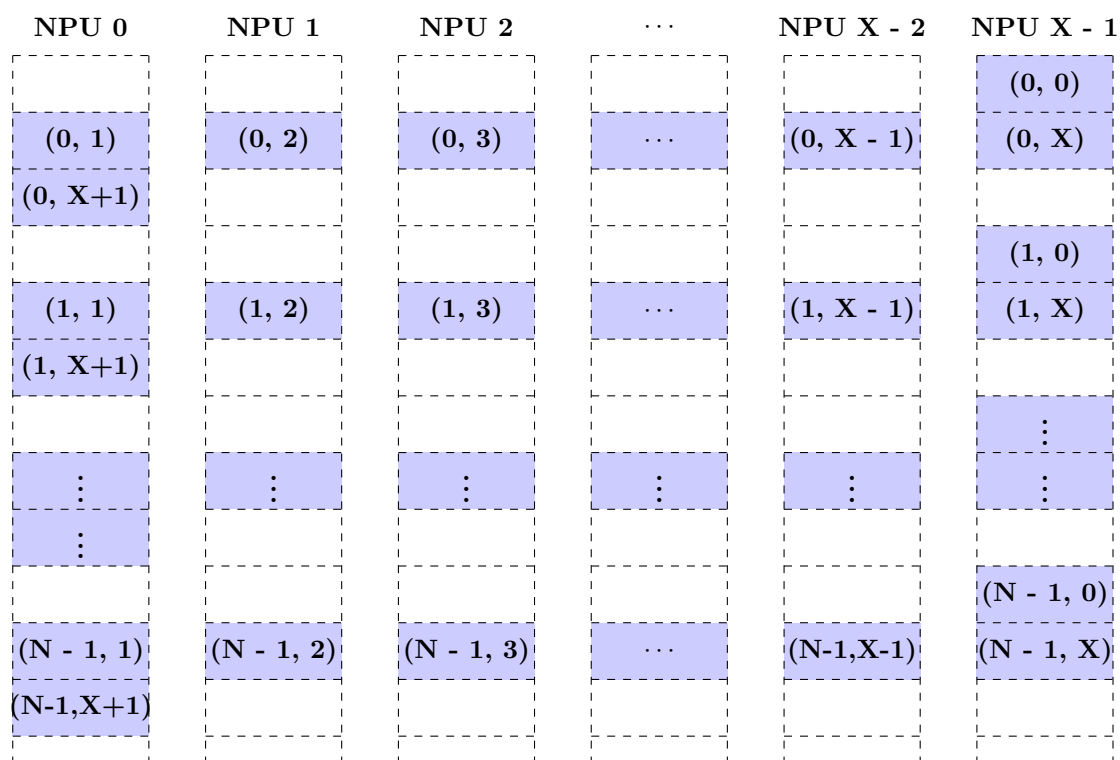
在 NPU0 的地址空间，没有数据分布。

情形 3: 当 Tensor 的 Shape 的维度 $C = X + 2$ ，当 Tensor 从 NPU0 开始存储时，那么 Tensor 在 Local Memory 上的排布方式如下所示。

NPU 0	NPU 1	NPU 2	...	NPU X - 2	NPU X - 1
(0, 0)	(0, 1)	(0, 2)	...	(0, X - 2)	(0, X - 1)
(0, X)	(0, X+1)				
(1, 0)	(1, 1)	(1, 2)	...	(1, X - 2)	(1, X - 1)
(1, X)	(1, X+1)				
⋮	⋮	⋮	⋮	⋮	⋮
⋮	⋮				
(N - 1, 0)	(N - 1, 1)	(N - 1, 2)	...	(N-1,X-2)	(N-1,X-1)
(N - 1, X)	(N-1,X+1)				

$C=X+1$ 维度的数据被排布到 NPU0，而 $C=X+2$ 维度的数据被排布到 NPU1。并且注意到，下一个 N 维度仍然从 NPU0 开始排布。

情形 4: 当 Tensor 的 Shape 的维度 $C = X + 2$ ，而 Tensor 从 NPU X-1 开始存储时，那么 Tensor 在 Local Memory 上的排布方式如下所示。



可以看到，C=0 维度的数据被排布到了 NPU X-1，C=1 维度的数据被排布到了 NPU0 上，依次类推，C= X+2 维度的数据被排布到了 NPU0 上。

Local Memory 上几种常用数据排布方式

上面介绍了 Tensor 在 Local Memory 上存储的基本原则，现在介绍 1684x 指令集常用的几种数据排布方式：

1. 64-Bytes 对齐存储方式

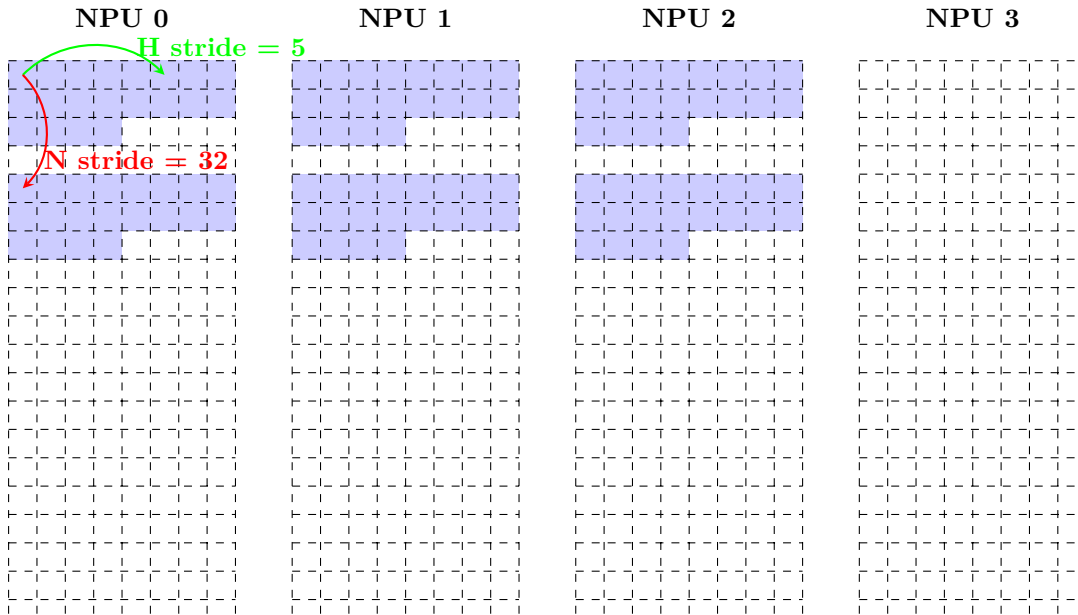
“64-Bytes 对齐存储方式”是最常用的 Tensor 存储方式，它是指 Tensor 排放存储要满足以下几个约束：

- Tensor 的起始地址是 64 的整数倍
- W_stride = 1
- H_stride = W
- C_stride = $\text{ceil}(H*W, 16) * 16$, 如果数据元素是 **32-bits** ; $\text{ceil}(H*W, 32) * 32$, 如果数据元素是 **16-bits** ; $\text{ceil}(H*W, 64) * 64$, 如果数据元素是 **8-bits**。
- N_stride = C_stride * (单个 NPU 上 channel 的个数)

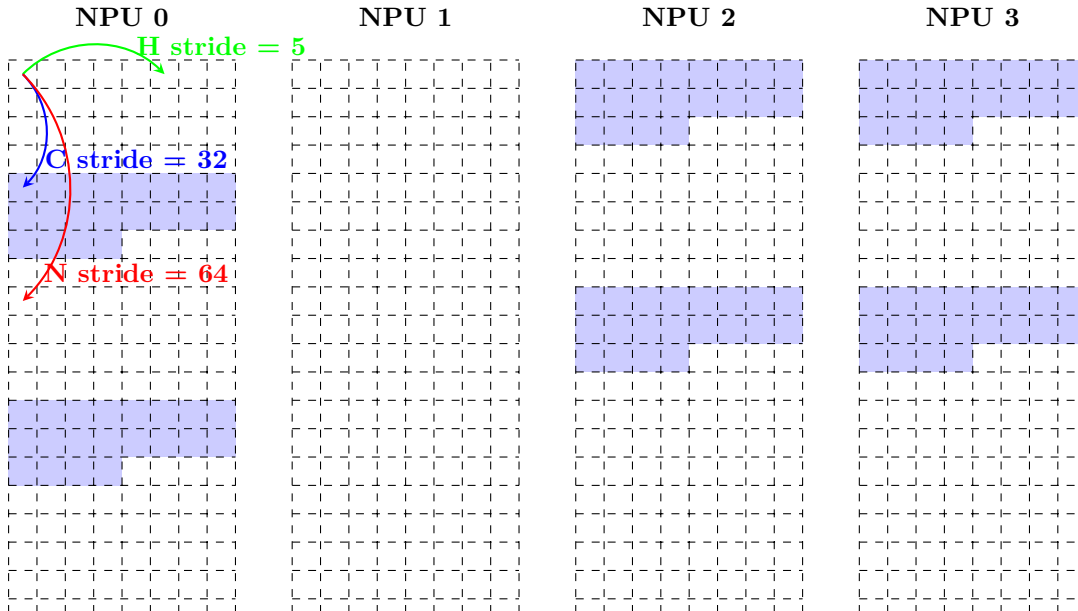
其中 ceil 是向上取整的意思。可通过 `tpu_aligned_stride()` 计算 stride。

为举例简便，假设 NPU 个数 =4

举例 1: Tensor 的 Shape(.N=2,.C=3,.H=4,.W=5), 数据类型为 float16, NPU0 开始存储



举例 2: Tensor 的 Shape(.N=2,.C=3,.H=4,.W=5), 数据类型为 float16, NPU2 开始存储



2. 紧凑存储方式

“紧凑存储方式”也是较为常用的 Tensor 存储方式。

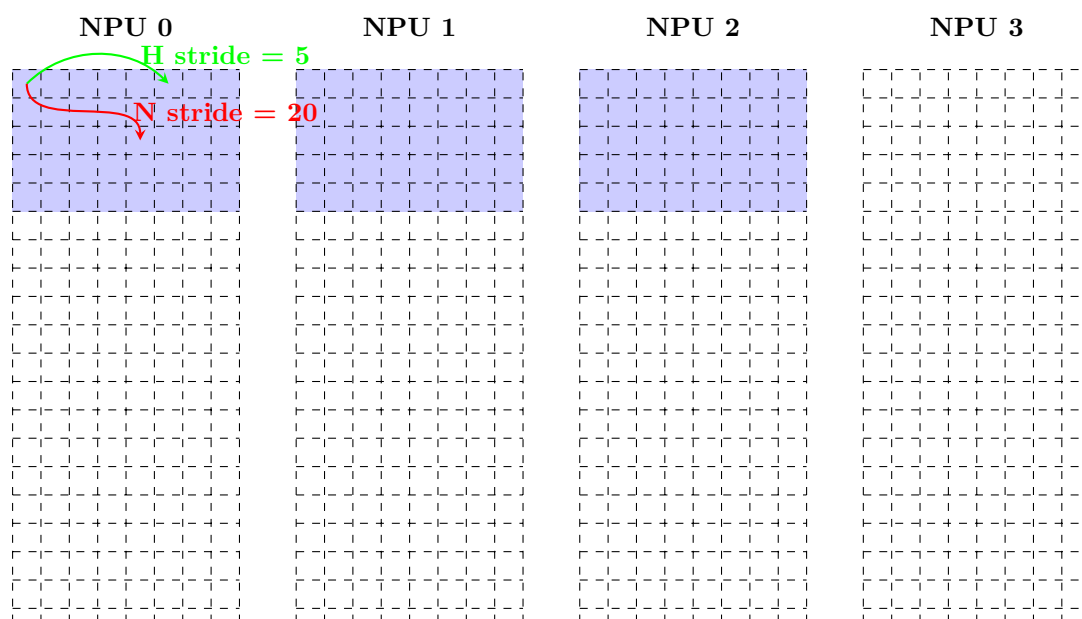
假设 Tensor 的 Shape 为 (N, C, H, W), 按照“紧凑存储方式存储”要满足以下约束:

- Tensor 的起始地址是 4 的整数倍。
- $W_stride = 1$
- $H_stride = W$
- $C_stride = H * W$
- $N_stride = C_stride * (\text{单个 NPU 上 channel 的个数})$

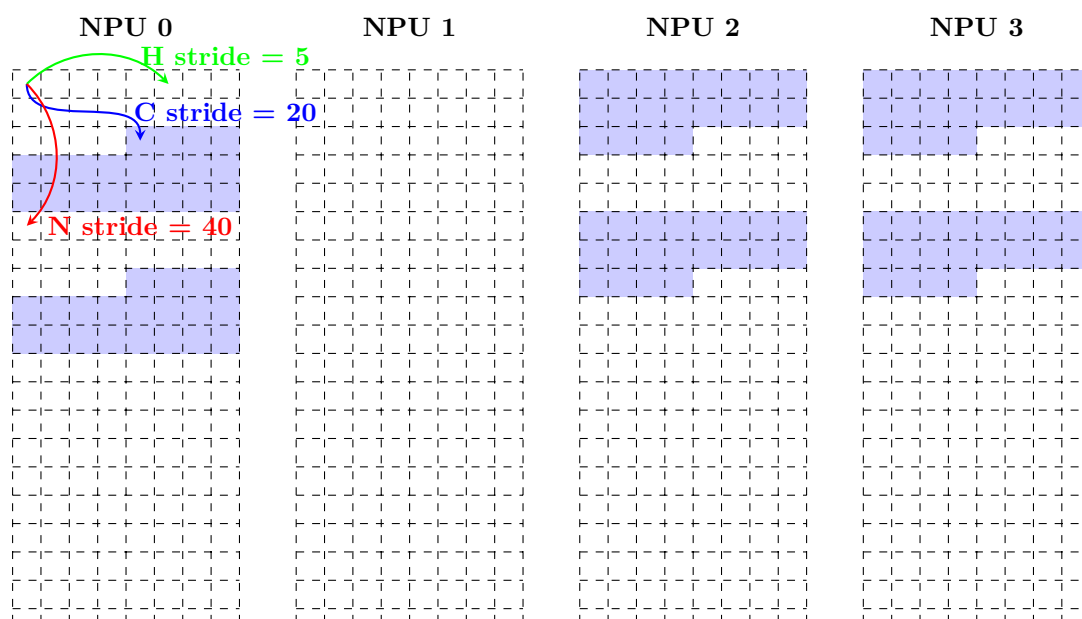
可通过 `tpu_compact_stride()` 计算 stride。

为举例简便, 假设 NPU 个数 = 4

举例 1: Tensor 的 Shape(.N=2,.C=3,.H=4,.W=5), 数据类型为 float16, NPU0 开始存储



举例 2: Tensor 的 Shape(.N=2,.C=3,.H=4,.W=5), 数据类型为 float16, NPU2 开始存储



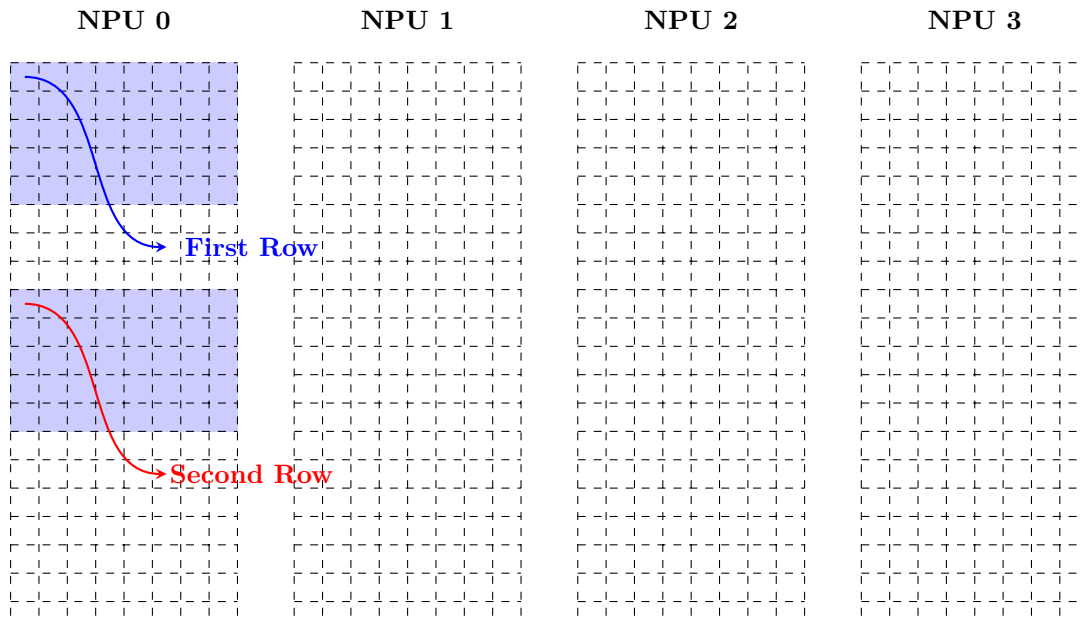
3. 矩阵存储方式

“矩阵存储方式”是矩阵运算指令用的数据存储方式。

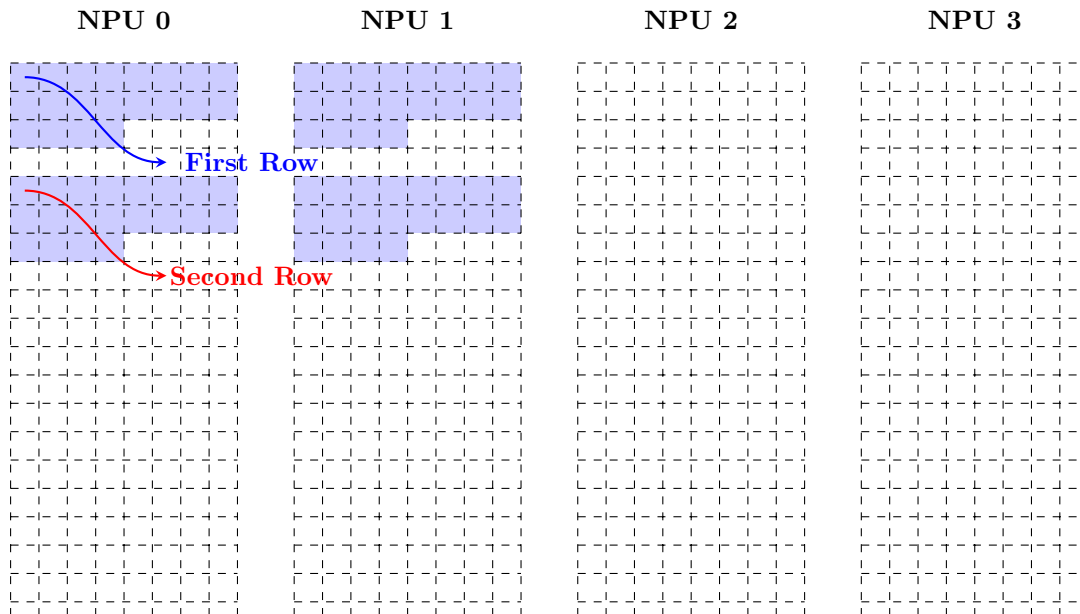
对于一个 $n \times m$ 的矩阵，可以用 Tensor 的 4 维数组的形式来进行表示，这个 Tensor 的 Shape 为 $(N=n, C=\text{ceil}(m/w), H=1, W=w)$ ，其中 w 可以为 $(1, m)$ 之间的任意值。

为举例简便，假设 NPU 个数 = 4

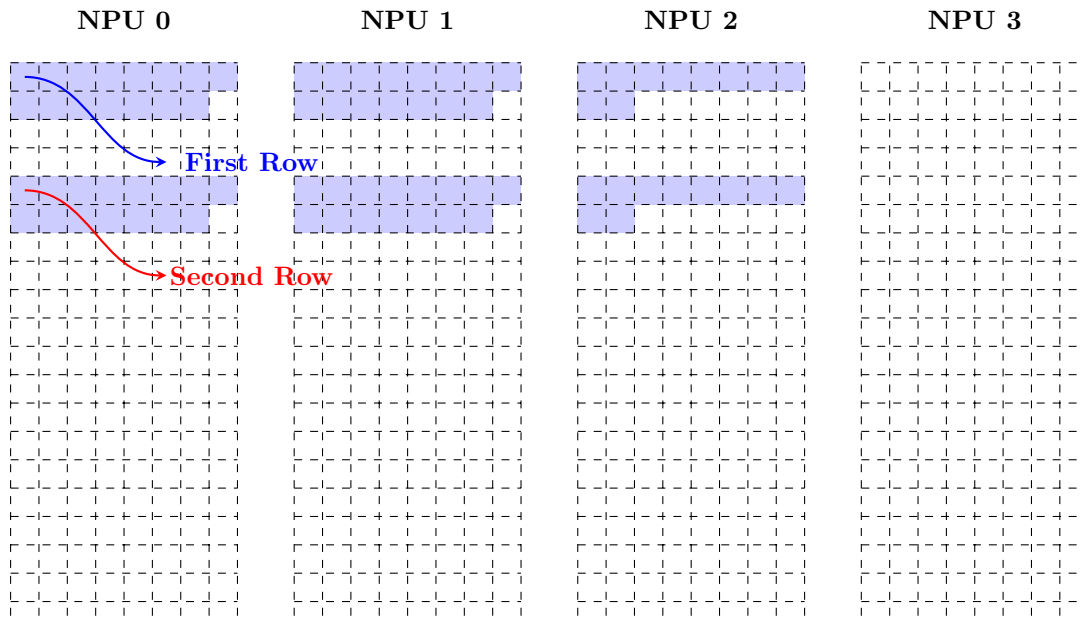
举例 1: 矩阵的形状为 (2×40) ，数据类型为 float16, $w = 40$



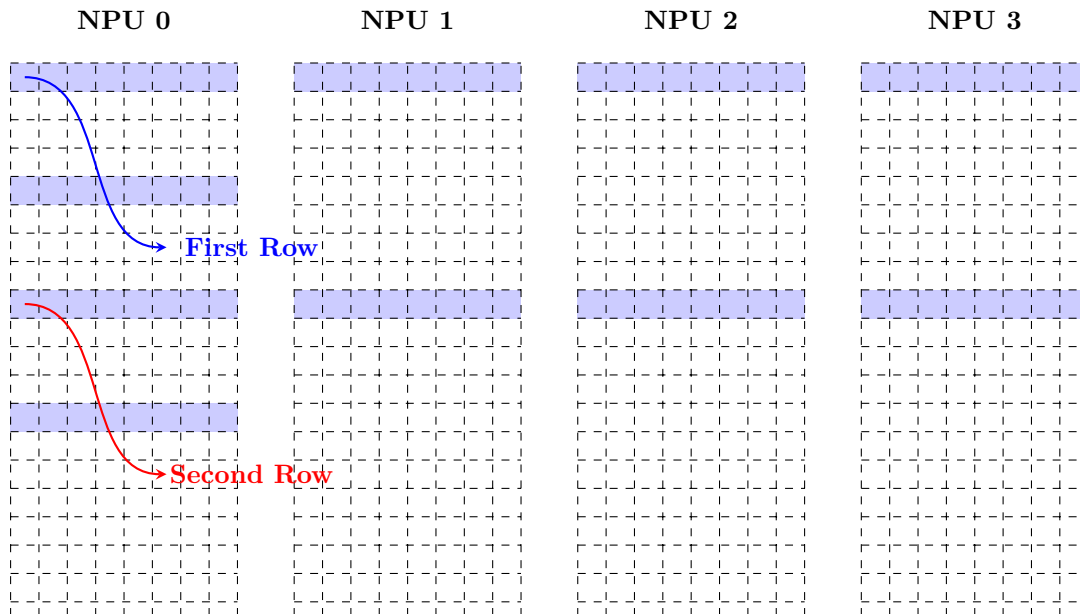
举例 2: 矩阵的形状为 (2x40), 数据类型为 float16, $w = 20$



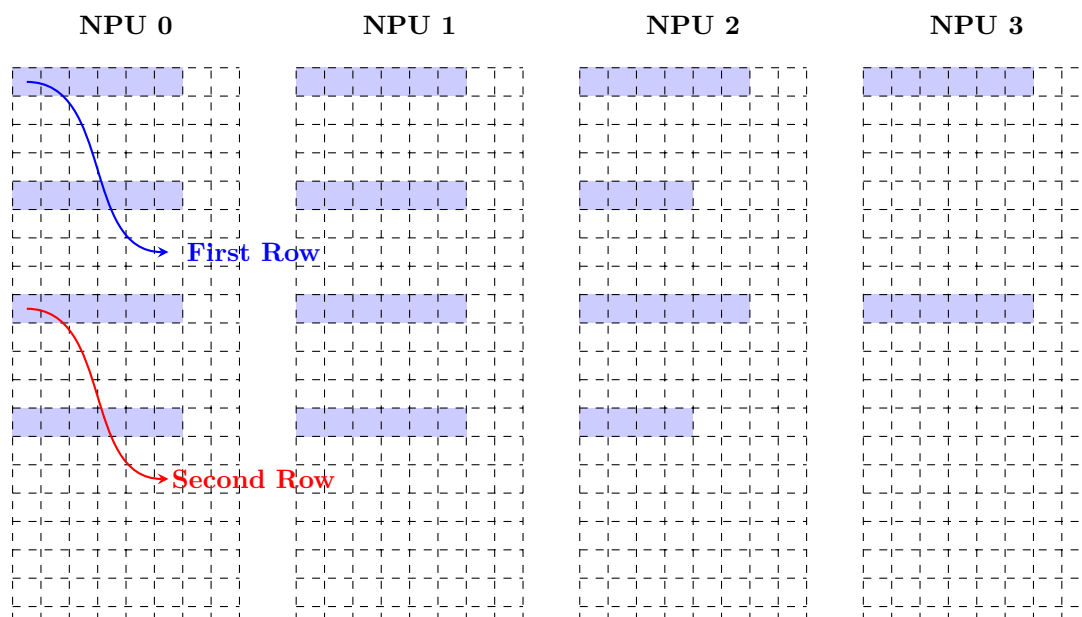
举例 3: 矩阵的形状为 (2x40), 数据类型为 float16, $w = 15$



举例 4: 矩阵的形状为 (2x40), 数据类型为 float16, $w = 8$



举例 5: 矩阵的形状为 (2x40), 数据类型为 float16, $w = 6$



4. 向量存储

对于一个 $1 \times m$ 的向量，可以用 Tensor 的 4 维数组的形式来进行表示，这个 Tensor 的 Shape 为 $(N=1, C=\text{ceil}(m/w), H=1, W=w)$ ，其中 w 可以为 $(1, m)$ 之间的任意值。

5. 行 64 字节对齐存储

一种 4D 张量在 local memory 中的存储格式。张量的 shape 是 (N, C, H, W) ，满足

- 地址被 64 整除
- W-stride 是 1
- H-stride 是 $\text{ceil}(W / 16) * 16$ ，如果元素的数据类型的位宽是 32-bit， $\text{ceil}(W / 32) * 32$ ，如果是 16-bit， $\text{ceil}(W / 64) * 64$ ，如果是 8-bit
- C-stride 是 $H * H\text{-stride}$
- N-stride 是 C-stride 乘以每个 NPU 的 channel 数

可通过 `tpu_line_aligned_stride()` 计算 stride。

6. 64IC/32IC 存储

64IC/32IC 存储是卷积核在 local memory 中的特殊存储方式，仅用于卷积计算过程。其中 INT8 kernel 以 64IC 格式存放，FP16/BFP16 kernel 以 32IC 格式存放。

假设卷积核的 shape 是 (ic, oc, kh, kw) ，分别表示 input channel、output channel、卷积核的高度以及卷积核的宽度。

64IC 存储满足：

- $W_stride = 64$
- $H_stride = 64 * kw$
- $C_stride = 64 * kw * kh * \text{ceil}(ic/64)$
- $N_stride = 64 * kw * kh * \text{ceil}(ic/64)$

其中，每 64 个 input channel 作为一组进行存储，每组之间的 stride 为 $64 * kw * kh$ 。

32IC 存储满足：

- $W_stride = 32$
- $H_stride = 32 * kw$
- $C_stride = 32 * kw * kh * \text{ceil}(ic/32)$
- $N_stride = 32 * kw * kh * \text{ceil}(ic/32)$

其中，每 32 个 input channel 作为一组进行存储，每组之间的 stride 为 $32 * kw * kh$ 。

4.1 基础定义

- 地址类型。

```
typedef unsigned int local_addr_t
```

```
typedef unsigned long long system_addr_t
```

```
typedef unsigned long long global_addr_t
```

```
typedef unsigned long long l2_sram_addr_t
```

```
typedef unsigned long long addr_t
```

- 数据维度。

```
class dim4
```

```
    int n
```

```
    int c
```

```
    int h
```

```
    int w
```

```
class dim2
```

```
int h
```

```
int w
```

- 数据类型。

```
enum data_type_t
```

```
enumerator DT_INT8 = 1
```

```
enumerator DT_UINT8 = 0
```

```
enumerator DT_INT16 = 7
```

```
enumerator DT_UINT16 = 6
```

```
enumerator DT_FP16 = 3
```

```
enumerator DT_BFP16 = 11
```

```
enumerator DT_INT32 = 9
```

```
enumerator DT_UINT32 = 8
```

```
enumerator DT_FP32 = 5
```

- 舍入模式枚举。

```
enum rounding_mode_t
```

```
enumerator RM_HALF_TO_EVEN = 0
```

```
enumerator RM_HALF_AWAY_FROM_ZERO = 1
```

```
enumerator RM_TOWARDS_ZERO = 2
```

```
enumerator RM_DOWN = 3
```

```
enumerator RM_UP = 4
```

```
enumerator RM_HALF_UP = 5
```

```
enumerator RM_HALF_DOWN = 6
```

- 半精度浮点格式。

```
union float16
```

```
unsigned short bits
```

- BF16 半精度浮点格式。

```
union bfloat16
```

```
unsigned short bits
```

-
- 标量联合体。

```
union scalar__t
```

```
char s8
```

```
unsigned char u8
```

```
short s16
```

```
unsigned short u16
```

```
float16 f16
```

```
bfloat16 bf16
```

```
int s32
```

```
unsigned int u32
```

```
float f32
```

-
- 变量类型枚举。

```
enum var__type__t
```

```
enumerator TENSOR
```

```
enumerator SCALAR
```

```
enumerator VECTOR
```

-
- 变量内容联合体。

```
union var_context_t
```

```
scalar_t scalar_t
```

```
local_addr_t addr
```

- 变量结构体，如果 `type` 是 `TENSOR` 或 `VECTOR`，则 `context.addr` 有效，否则 `context.scalar` 有效。

```
class variable_t
```

```
var_type_t type
```

```
var_context_t context
```

- NPU 的数量。

```
NPU_NUM
```

- 每个 NPU 上 local memory 的大小。

```
LOCAL_MEM_SIZE
```

- 基础数学运算。

```
DIV_UP
```

```
DIV_UP(a, b) (((a) - 1) / (b) + 1)
```

```
ALIGN
```

```
ALIGN(a, b) DIV_UP (a, b) * (b)
```

4.2 舍入模式

舍入是指按照一定的规则舍去某些数字后面多余的尾数的过程，以得到更简短、明确的数字表示。给定 x ，舍入结果是 y ，有下面的舍入模式供选择。

4.2.1 邻近偶数四舍五入 (Half to Even)

四舍五入，当小数值为 0.5 时舍入到邻近的偶数，对应的枚举值是 `RM_HALF_TO_EVEN`。

4.2.2 远离原点四舍五入 (Half Away From Zero)

四舍五入，正数接近于正无穷，负数接近于负无穷，对应的枚举值是RM_HALF_AWAY_FROM_ZERO，公式如下

$$y = \text{sign}(x) \lfloor |x| + 0.5 \rfloor = -\text{sign}(x) \lceil -|x| - 0.5 \rceil$$

4.2.3 截断取整 (Towards Zero)

无条件舍去，接近于原点，对应的枚举值是RM_TOWARDS_ZERO，公式如下

$$y = \text{sign}(x) \lfloor |x| \rfloor = -\text{sign}(x) \lceil -|x| \rceil = \begin{cases} \lfloor x \rfloor & \text{if } x > 0, \\ \lceil x \rceil & \text{otherwise.} \end{cases}$$

4.2.4 下取整 (Down)

接近于负无穷，对应的枚举值是RM_DOWN，公式如下

$$y = \lfloor x \rfloor = -\lceil -x \rceil$$

4.2.5 上取整 (Up)

接近于正无穷，对应的枚举值是RM_UP，公式如下

$$y = \lceil x \rceil = -\lfloor -x \rfloor$$

4.2.6 向上四舍五入 (Half Up)

四舍五入，接近于正无穷，对应的枚举值是RM_HALF_UP，公式如下

$$y = \lceil x + 0.5 \rceil = -\lfloor -x - 0.5 \rfloor = \left\lceil \frac{\lfloor 2x \rfloor}{2} \right\rceil$$

4.2.7 向下四舍五入 (Half Down)

四舍五入，接近于正无穷，对应的枚举值是RM_HALF_DOWN，公式如下

$$y = \lfloor x - 0.5 \rfloor = -\lceil -x + 0.5 \rceil = \left\lfloor \frac{\lceil 2x \rceil}{2} \right\rfloor$$

4.2.8 例子

下表列出不同舍入模式下 x 与 y 的对应关系。

	Half to Even	Half Away From Zero	Towards Zero	Down	Up	Half Up	Half Down
+1.8	+2	+2	+1	+1	+2	+2	+2
+1.5	+2	+2	+1	+1	+2	+2	+1
+1.2	+1	+1	+1	+1	+2	+1	+1
+0.8	+1	+1	0	0	+1	+1	+1
+0.5	0	+1	0	0	+1	+1	0
+0.2	0	0	0	0	+1	0	0
-0.2	0	0	0	-1	0	0	0
-0.5	0	-1	0	-1	0	0	-1
-0.8	-1	-1	0	-1	0	-1	-1
-1.2	-1	-1	-1	-2	-1	-1	-1
-1.5	-2	-2	-1	-2	-1	-1	-2
-1.8	-2	-2	-1	-2	-1	-2	-2

4.3 功能函数

4.3.1 tpu_initialize

在使用 GDMA 和 BDC 操作前初始化设备。

```
void tpu_initialize()
```

4.3.2 tpu_poll

同步设备直到之前下发的 GDMA 和 BDC 操作结束返回。

```
void tpu_poll()
```

注意事项

- 调用此方法前，GDMA 和 BDC 须处于非并行状态。

4.3.3 tpu_parallel_start

开启 GDMA 和 BDC 的并行状态。

```
void tpu_parallel_start()
```

注意事项

- 调用此方法前，GDMA 和 BDC 须处于非并行状态。

4.3.4 tpu_parallel_end

结束 GDMA 和 BDC 的并行状态。

```
void tpu_parallel_end()
```

注意事项

- 调用此方法前，GDMA 和 BDC 须处于并行状态。

4.3.5 tpu_is_parallel_state

查看 GDMA 和 BDC 的并行状态。

```
bool tpu_is_parallel_state()
```

返回 并行状态标志

4.3.6 tpu_npu_num

每个 TPU 中 NPU 的数量。

```
int tpu_npu_num()
```

返回 NPU 的数量

4.3.7 tpu_bank_num

local memory 中 Bank 的数量。

```
int tpu_bank_num()
```

返回 Bank 的数量

4.3.8 tpu_eu_num

不同数据类型对应的 EU 的数量。

```
int tpu_eu_num(data_type_t dtype)
```

参数 dtype—数据类型

返回 EU 的数量

4.3.9 tpu_local_mem_size_per_npu

每个 NPU 的 local memory 的大小（以字节为单位）。

```
int tpu_local_mem_size_per_npu()
```

返回 local memory 的大小

4.3.10 tpu_l2_sram_size

L2-SRAM 的大小（以字节为单位）。

```
int tpu_l2_sram_size()
```

返回 L2-SRAM 的大小

4.3.11 tpu_l2_sram_get_start_addr

L2-SRAM 起始地址。

```
unsigned long long tpu_l2_sram_get_start_addr()
```

返回 L2-SRAM 的起始地址

4.3.12 tpu_local_mem_get_start_addr

第一个 local memory 的起始地址。

```
unsigned int tpu_local_mem_get_start_addr()
```

返回 第一个 local memory 的起始地址

4.3.13 tpu_global_mem_addr

通过 global memory 的地址得到指针

```
void *tpu_global_mem_addr(global_addr_t addr)
```

参数 addr -global memory 的地址

返回 指向 global memory 的指针

4.3.14 tpu_local_mem_addr

通过 NPU 的 index 和 local memory 的地址得到指针

```
void *tpu_local_mem_addr(int start_idx, local_addr_t addr)
```

参数

- start_idx -起始 NPU 的 index
- addr -local memory 的地址

返回 指向 local memory 的指针

注意事项

- start_idx 的取值范围是 [0, NPU_NUM - 1]。
- addr 的取值范围是 [0, LOCAL_MEM_SIZE - 1]。

4.3.15 tpu_local_mem_addr_unified

通过统一的 local memory 的地址得到指针

```
void *tpu_local_mem_addr_unified(local_addr_t addr)
```

参数 addr –local memory 的地址

返回 指向 local memory 的指针

注意事项

- addr 的取值范围是 $[0, \text{NPU_NUM} * \text{LOCAL_MEM_SIZE} - 1]$ 。
- 等价于 `tpu_local_mem_addr (tpu_npu_index(addr) , addr % LOCAL_MEM_SIZE)`。

4.3.16 tpu_l2_sram_addr

通过 L2-SRAM 的地址得到指针

```
void *tpu_l2_sram_addr(l2_sram_addr_t addr)
```

参数 addr –L2-SRAM 的地址

返回 指向 L2-SRAM 的指针

4.3.17 tpu_flush_cache

将当前 cache 的数据写回到 DDR

```
void tpu_flush_cache(system_addr_t address, unsigned long long size)
```

参数

- address –DDR 地址, 64 字节对齐
- size –刷新的数据大小, 64 字节的倍数

4.3.18 tpu_invalidate_cache

使当前 cache 无效, 数据直接从 DDR 中获取

```
void tpu_invalidate_cache(system_addr_t address, unsigned long long size)
```

参数

- address –DDR 地址, 64 字节对齐
- size –数据大小, 64 字节的倍数

4.4 数据同步函数

4.4.1 tpu_bdc_send_msg

BDC 引擎发送消息到消息队列对应的索引中，用于数据同步。 注意事项

- BM1684X 设备不支持该功能。
- msg_id 最大值是 127。
- wait_cnt 最大值为 7。

4.4.2 tpu_bdc_wait_msg

BDC 引擎等待指定的消息索引中发送的消息，只有等到的消息数量等于指定的数量后该引擎才能继续往下执行，用于数据同步。 注意事项

- BM1684X 设备不支持该功能。
- msg_id 最大值是 127。
- send_cnt 最大值为 7。

4.4.3 tpu_gdma_send_msg

GDMA 引擎发送消息到消息队列对应的索引中，用于数据同步。 注意事项

- BM1684X 设备不支持该功能。
- msg_id 最大值是 127。
- wait_cnt 最大值为 7。

4.4.4 tpu_gdma_wait_msg

GDMA 引擎等待指定的消息索引中发送的消息，只有等到的消息数量等于指定的数量后该引擎才能继续往下执行，用于数据同步。 注意事项

- BM1684X 设备不支持该功能。
- msg_id 最大值是 127。
- send_cnt 最大值为 7。

4.4.5 tpu_hau_send_msg

HAU 引擎发送消息到消息队列对应的索引中，用于数据同步。 注意事项

- BM1684X 设备不支持该功能。
- msg_id 最大值是 127。
- wait_cnt 最大值为 7。

4.4.6 tpu_hau_wait_msg

HAU 引擎等待指定的消息索引中发送的消息，只有等到的消息数量等于指定的数量后该引擎才能继续往下执行，用于数

注意事项

- BM1684X 设备不支持该功能。
- msg_id 最大值是 127。
- send_cnt 最大值为 7。

4.5 辅助函数

4.5.1 tpu_data_type_size

获取某数据类型所占的空间大小。

```
int tpu_data_type_size(data_type_t dtype)
```

** 不支持 DT_INT4/DT_UINT4, 在该类型下会报 ASSERT

4.5.2 tpu_data_type_bits

获取某数据类型比特位宽。

```
int tpu_data_type_bits(data_type_t dtype)
```

4.5.3 tpu_npu_index

获取某地址对应的 NPU 的索引。

```
int tpu_npu_index(local_addr_t addr)
```

4.5.4 tpu_bank_index

获取某地址对应的 BANK 索引。

```
int tpu_bank_index(local_addr_t addr)
```

4.5.5 tpu_channle_num_per_npu

获取 tensor 在 NPU 中分配的 channel 数量。

```
int tpu_channle_num_per_npu(int start_idx, int num_channels)
```

参数

- start_idx - 起始 NPU 的 index
- num_channels - tensor 的 channel 的数量

返回 NPU 分配的 channel 数量

4.5.6 tpu_aligned_feature_size

以 64-Byte 为存储单元进行内存分配时，计算输入二维向量需要分配的存储单元数量。

```
int tpu_aligned_feature_size(int h, int w, data_type_t dtype)
```

参数

- h - 二维向量的高度 h
- w - 二维向量的宽度 w
- dtype - 元素的数据类型

4.5.7 tpu_aligned_stride

以 64-Bytes 对齐存储方式计算输入 tensor 的 stride。

```
void tpu_aligned_stride(dim4 *stride, int start_idx, const dim4 *shape, data_type_t dtype)
```

参数

- stride - 指向 stride 的指针
- start_idx - 起始 NPU 的 index
- shape - 指向 shape 的指针
- dtype - 元素的数据类型

4.5.8 tpu_compact_stride

以紧凑存储方式计算输入 tensor 的 stride。

```
void tpu_compact_stride(dim4 *stride, int start_idx, const dim4 *shape)
```

参数

- stride - 指向 stride 的指针
- start_idx - 起始 NPU 的 index
- shape - 指向 shape 的指针

4.5.9 tpu_line_aligned_stride

以行 64 字节对齐存储计算输入 tensor 的 stride。

```
void tpu_line_aligned_stride(dim4 *stride, int start_idx, const dim4 *shape, data_type_t dtype)
```

参数

- stride - 指向 stride 的指针
- start_idx - 起始 NPU 的 index
- shape - 指向 shape 的指针
- dtype - 元素的数据类型

4.5.10 tpu_continuous_stride

以连续存储方式计算输入 tensor 的 stride。

```
void tpu_continuous_stride(dim4 *stride, const dim4 *shape)
```

参数

- stride –指向 stride 的指针
- shape –指向 shape 的指针
- dtype –元素的数据类型

4.6 GDMA 操作

4.6.1 tpu_gdma_general_cpy_S2L

张量的元素从 system memory 拷贝到 local memory。

```
void tpu_gdma_general_cpy_S2L(local_addr_t dst_addr, system_addr_t src_addr,
                               const dim4 *dst_shape, const dim4 *src_shape, const
                               dim4 *dst_stride, const dim4 *src_stride,
                               data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(n, c, h, w)$$

参数

- dst_addr –dst 在 local memory 中的地址
- src_addr –src 在 system memory 中的地址
- dst_shape –指向 dst 的 shape 的指针
- src_shape –指向 src 的 shape 的指针
- dst_stride –指向 dst 的 stride 的指针
- src_stride –指向 src 的 stride 的指针
- dtype –dst 和 src 的元素的数据类型

注意事项

- dst_shape 和 src_shape 对应的元素个数应相等。
- 如果 dst_stride 是 NULL，则 dst 是 64-byte aligned layout，否则是 free layout。
- 如果 src_stride 是 NULL，则 src 是 continuous layout，否则是 free layout。
- dst_stride->w 和 src_stride->w 小于等于 128 / tpu_data_type_size(dtype)。

4.6.2 tpu_gdma_general_cpy_L2S

张量的元素从 local memory 拷贝到 system memory。

```
void tpu_gdma_general_cpy_L2S(system_addr_t dst_addr, local_addr_t src_addr,
                              const dim4 *dst_shape, const dim4 *src_shape, const
                              dim4 *dst_stride, const dim4 *src_stride,
                              data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(n, c, h, w)$$

参数

- dst_addr -dst 在 system memory 中的地址
- src_addr -src 在 local memory 中的地址
- dst_shape -指向 dst 的 shape 的指针
- src_shape -指向 src 的 shape 的指针
- dst_stride -指向 dst 的 stride 的指针
- src_stride -指向 src 的 stride 的指针
- dtype -dst 和 src 的元素的数据类型

注意事项

- dst_shape 和 src_shape 对应的元素个数应相等。
- 如果 dst_stride 是 NULL, 则 dst 是 continuous layout, 否则是 free layout。
- 如果 src_stride 是 NULL, 则 src 是 64-byte aligned layout, 否则是 free layout。
- dst_stride->w 和 src_stride->w 小于等于 128 / tpu_data_type_size(dtype)。

4.6.3 tpu_gdma_cpy_S2L

张量的元素从 system memory 拷贝到 local memory。

```
void tpu_gdma_cpy_S2L(local_addr_t dst_addr, system_addr_t src_addr, const dim4
                      *shape, const dim4 *dst_stride, const dim4 *src_stride,
                      data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(n, c, h, w)$$

参数

- dst_addr -dst 在 local memory 中的地址
- src_addr -src 在 system memory 中的地址
- shape -指向 dst 和 src 的 shape 的指针
- dst_stride -指向 dst 的 stride 的指针
- src_stride -指向 src 的 stride 的指针
- dtype -dst 和 src 的元素的数据类型

注意事项

- 如果 dst_stride 是 NULL, 则 dst 是 64-byte aligned layout, 否则是 free layout。
- 如果 src_stride 是 NULL, 则 src 是 continuous layout, 否则是 free layout。
- dst_stride->w 和 src_stride->w 小于等于 128 / tpu_data_type_size(dtype)。

4.6.4 tpu_gdma_cpy_L2S

张量的元素从 local memory 拷贝到 system memory。

```
void tpu_gdma_cpy_L2S(system_addr_t dst_addr, local_addr_t src_addr, const dim4
                      *shape, const dim4 *dst_stride, const dim4 *src_stride,
                      data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(n, c, h, w)$$

参数

- dst_addr -dst 在 system memory 中的地址
- src_addr -src 在 local memory 中的地址
- shape -指向 dst 和 src 的 shape 的指针
- dst_stride -指向 dst 的 stride 的指针
- src_stride -指向 src 的 stride 的指针
- dtype -dst 和 src 的元素的数据类型

注意事项

- 如果 dst_stride 是 NULL, 则 dst 是 continuous layout, 否则是 free layout。
- 如果 src_stride 是 NULL, 则 src 是 64-byte aligned layout, 否则是 free layout。
- dst_stride->w 和 src_stride->w 小于等于 128 / tpu_data_type_size(dtype)。

4.6.5 tpu_gdma_cpy_L2L

张量的元素从 local memory 拷贝到 local memory。

```
void tpu_gdma_cpy_L2L(local_addr_t dst_addr, local_addr_t src_addr, const dim4
                      *shape, const dim4 *dst_stride, const dim4 *src_stride,
                      data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(n, c, h, w)$$

参数

- dst_addr -dst 在 local memory 中的地址
- src_addr -src 在 local memory 中的地址
- shape -指向 dst 和 src 的 shape 的指针
- dst_stride -指向 dst 的 stride 的指针
- src_stride -指向 src 的 stride 的指针
- dtype -dst 和 src 的元素的数据类型

注意事项

- 如果 dst_stride 是 NULL, 则 dst 是 64-byte aligned layout, 否则是 free layout。
- 如果 src_stride 是 NULL, 则 src 是 64-byte aligned layout, 否则是 free layout。
- dst_stride->w 和 src_stride->w 小于等于 128 / tpu_data_type_size(dtype)。

4.6.6 tpu_gdma_cpy_S2S

张量的元素从 system memory 拷贝到 system memory。

```
void tpu_gdma_cpy_S2S(system_addr_t dst_addr, system_addr_t src_addr, const
    dim4 *shape, const dim4 *dst_stride, const dim4 *src_stride,
    data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(n, c, h, w)$$

参数

- dst_addr -dst 在 system memory 中的地址
- src_addr -src 在 system memory 中的地址
- shape -指向 dst 和 src 的 shape 的指针
- dst_stride -指向 dst 的 stride 的指针
- src_stride -指向 src 的 stride 的指针
- dtype -dst 和 src 的元素的数据类型

注意事项

- 如果 dst_stride 是 NULL, 则 dst 是 continuous layout, 否则是 free layout。
- 如果 src_stride 是 NULL, 则 src 是 continuous layout, 否则是 free layout。
- dst_stride->w 和 src_stride->w 小于等于 128 / tpu_data_type_size(dtype)。

4.6.7 tpu_gdma_cpy_nc_trans_S2L

张量的元素从 system memory 拷贝到 local memory, N 和 C 维度转置。

```
void tpu_gdma_cpy_nc_trans_S2L(local_addr_t dst_addr, system_addr_t src_addr,
    const dim4 *dst_shape, const dim4 *dst_stride,
    const dim4 *src_stride, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(c, n, h, w)$$

参数

- dst_addr -dst 在 local memory 中的地址
- src_addr -src 在 system memory 中的地址
- dst_shape -指向 dst 的 shape 的指针
- dst_stride -指向 dst 的 stride 的指针
- src_stride -指向 src 的 stride 的指针
- dtype -dst 和 src 的元素的数据类型

注意事项

- src 的 shape 是 [dst_shape->c, dst_shape->n, dst_shape->h, dst_shape->w]。
- 如果 dst_stride 是 NULL, 则 dst 是 64-byte aligned layout, 否则是 free layout。
- 如果 src_stride 是 NULL, 则 src 是 continuous layout, 否则是 free layout。
- dst_stride->w 和 src_stride->w 小于等于 128 / tpu_data_type_size(dtype)。

4.6.8 tpu_gdma_cpy_nc_trans_L2S

张量的元素从 local memory 拷贝到 system memory, N 和 C 维度转置。

```
void tpu_gdma_cpy_nc_trans_L2S(system_addr_t dst_addr, local_addr_t src_addr,
                               const dim4 *dst_shape, const dim4 *dst_stride,
                               const dim4 *src_stride, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(c, n, h, w)$$

参数

- `dst_addr` -dst 在 system memory 中的地址
- `src_addr` -src 在 local memory 中的地址
- `dst_shape` -指向 dst 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针
- `src_stride` -指向 src 的 stride 的指针
- `dtype` -dst 和 src 的元素的数据类型

注意事项

- src 的 shape 是 `[dst_shape->c, dst_shape->n, dst_shape->h, dst_shape->w]`。
- 如果 `dst_stride` 是 NULL, 则 dst 是 continuous layout, 否则是 free layout。
- 如果 `src_stride` 是 NULL, 则 src 是 64-byte aligned layout, 否则是 free layout。
- `dst_stride->w` 和 `src_stride->w` 小于等于 `128 / tpu_data_type_size(dtype)`。

4.6.9 tpu_gdma_cpy_nc_trans_L2L

张量的元素从 local memory 拷贝到 local memory, N 和 C 维度转置。

```
void tpu_gdma_cpy_nc_trans_L2L(local_addr_t dst_addr, local_addr_t src_addr,
                               const dim4 *dst_shape, const dim4 *dst_stride,
                               const dim4 *src_stride, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(c, n, h, w)$$

参数

- `dst_addr` -dst 在 local memory 中的地址
- `src_addr` -src 在 local memory 中的地址
- `dst_shape` -指向 dst 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针
- `src_stride` -指向 src 的 stride 的指针
- `dtype` -dst 和 src 的元素的数据类型

注意事项

- src 的 shape 是 `[dst_shape->c, dst_shape->n, dst_shape->h, dst_shape->w]`。
- 如果 `dst_stride` 是 NULL, 则 dst 是 64-byte aligned layout, 否则是 free layout。
- 如果 `src_stride` 是 NULL, 则 src 是 64-byte aligned layout, 否则是 free layout。
- `dst_stride->w` 和 `src_stride->w` 小于等于 `128 / tpu_data_type_size(dtype)`。

4.6.10 tpu_gdma_cpy_nc_trans_S2S

张量的元素从 system memory 拷贝到 system memory, N 和 C 维度转置。

```
void tpu_gdma_cpy_nc_trans_S2S(system_addr_t dst_addr, system_addr_t src_addr,
                               const dim4 *dst_shape, const dim4 *dst_stride,
                               const dim4 *src_stride, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(c, n, h, w)$$

参数

- `dst_addr` -dst 在 system memory 中的地址
- `src_addr` -src 在 system memory 中的地址
- `dst_shape` -指向 dst 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针
- `src_stride` -指向 src 的 stride 的指针
- `dtype` -dst 和 src 的元素的数据类型

注意事项

- src 的 shape 是 `[dst_shape->c, dst_shape->n, dst_shape->h, dst_shape->w]`。
- 如果 `dst_stride` 是 NULL, 则 dst 是 continuous layout, 否则是 free layout。
- 如果 `src_stride` 是 NULL, 则 src 是 continuous layout, 否则是 free layout。
- `dst_stride->w` 和 `src_stride->w` 小于等于 `128 / tpu_data_type_size(dtype)`。

4.6.11 tpu_gdma_cpy_cw_trans_S2L

张量的元素从 system memory 拷贝到 local memory, C 和 W 维度转置。

```
void tpu_gdma_cpy_cw_trans_S2L(local_addr_t dst_addr, system_addr_t src_addr,
                               const dim4 *dst_shape, const dim4 *dst_stride,
                               const dim4 *src_stride, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(n, w, h, c)$$

参数

- `dst_addr` -dst 在 local memory 中的地址
- `src_addr` -src 在 system memory 中的地址
- `dst_shape` -指向 dst 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针
- `src_stride` -指向 src 的 stride 的指针
- `dtype` -dst 和 src 的元素的数据类型

注意事项

- src 的 shape 是 `[dst_shape->n, dst_shape->w, dst_shape->h, dst_shape->c]`。
- 如果 `dst_stride` 是 NULL, 则 dst 是 64-byte aligned layout, 否则是 free layout。
- 如果 `src_stride` 是 NULL, 则 src 是 continuous layout, 否则是 free layout。
- `dst_stride->w` 和 `src_stride->w` 只能是 1。

4.6.12 tpu_gdma_cpy_cw_trans_L2S

张量的元素从 local memory 拷贝到 system memory, C 和 W 维度转置。

```
void tpu_gdma_cpy_cw_trans_L2S(system_addr_t dst_addr, local_addr_t src_addr,
                               const dim4 *dst_shape, const dim4 *dst_stride,
                               const dim4 *src_stride, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(n, w, h, c)$$

参数

- dst_addr -dst 在 system memory 中的地址
- src_addr -src 在 local memory 中的地址
- dst_shape -指向 dst 的 shape 的指针
- dst_stride -指向 dst 的 stride 的指针
- src_stride -指向 src 的 stride 的指针
- dtype -dst 和 src 的元素的数据类型

注意事项

- src 的 shape 是 [dst_shape->n, dst_shape->w, dst_shape->h, dst_shape->c]。
- 如果 dst_stride 是 NULL, 则 dst 是 continuous layout, 否则是 free layout。
- 如果 src_stride 是 NULL, 则 src 是 64-byte aligned layout, 否则是 free layout。
- dst_stride->w 和 src_stride->w 只能是 1。

4.6.13 tpu_gdma_cpy_cw_trans_L2L

张量的元素从 local memory 拷贝到 local memory, C 和 W 维度转置。

```
void tpu_gdma_cpy_cw_trans_L2L(local_addr_t dst_addr, local_addr_t src_addr,
                               const dim4 *dst_shape, const dim4 *dst_stride,
                               const dim4 *src_stride, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(n, w, h, c)$$

参数

- dst_addr -dst 在 local memory 中的地址
- src_addr -src 在 local memory 中的地址
- dst_shape -指向 dst 的 shape 的指针
- dst_stride -指向 dst 的 stride 的指针
- src_stride -指向 src 的 stride 的指针
- dtype -dst 和 src 的元素的数据类型

注意事项

- src 的 shape 是 [dst_shape->n, dst_shape->w, dst_shape->h, dst_shape->c]。
- 如果 dst_stride 是 NULL, 则 dst 是 64-byte aligned layout, 否则是 free layout。
- 如果 src_stride 是 NULL, 则 src 是 64-byte aligned layout, 否则是 free layout。
- dst_stride->w 和 src_stride->w 只能是 1。

4.6.14 tpu_gdma_cpy_cw_trans_S2S

张量的元素从 system memory 拷贝到 system memory, C 和 W 维度转置。

```
void tpu_gdma_cpy_cw_trans_S2S(system_addr_t dst_addr, system_addr_t
                               src_addr, const dim4 *dst_shape, const dim4
                               *dst_stride, const dim4 *src_stride, data_type_t
                               dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(n, w, h, c)$$

参数

- dst_addr -dst 在 system memory 中的地址
- src_addr -src 在 system memory 中的地址
- dst_shape -指向 dst 的 shape 的指针
- dst_stride -指向 dst 的 stride 的指针
- src_stride -指向 src 的 stride 的指针
- dtype -dst 和 src 的元素的数据类型

注意事项

- src 的 shape 是 [dst_shape->n, dst_shape->w, dst_shape->h, dst_shape->c]。
- 如果 dst_stride 是 NULL, 则 dst 是 continuous layout, 否则是 free layout。
- 如果 src_stride 是 NULL, 则 src 是 continuous layout, 否则是 free layout。
- dst_stride->w 和 src_stride->w 只能是 1。

4.6.15 tpu_gdma_mask_select_L2S

将 local memory 中存储的张量按照 mask 筛选后拷贝到 global memory 中。

```
void tpu_gdma_mask_select_L2S(global_addr_t dst_addr, local_addr_t src_addr,
                               addr_t mask_addr, int mask_in_lmem, const dim4
                               *shape, data_type_t data_dtype, data_type_t
                               mask_dtype)
```

参数

- dst_addr -dst 在 global memory 中的地址
- src_addr -src 在 local memory 中的地址
- mask_addr -mask 在 global memory 或 local memory 中的地址
- mask_in_lmem -mask 在 local memory 的标志
- shape -指向 input data 或 mask 的 shape 的指针
- data_dtype -input/output data 的数据类型
- mask_dtype -mask 的数据类型

注意事项

- mask_select 后的 filter_num 可由 tpu_gdma_get_filter_num() 得到。该函数没有输入参数, 返回值的类型为 DT_UINT32。

4.6.16 tpu_gdma_mask_select_S2S

将 global memory 中存储的张量按照 mask 筛选后拷贝到 global memory 中。

```
void tpu_gdma_mask_select_L2S(global_addr_t dst_addr, global_addr_t src_addr,
                             addr_t mask_addr, int mask_in_lmem, const dim4
                             *shape, data_type_t data_dtype, data_type_t
                             mask_dtype)
```

参数

- dst_addr -dst 在 global memory 中的地址
- src_addr -src 在 global memory 中的地址
- mask_addr -mask 在 global memory 或 local memory 中的地址
- mask_in_lmem -mask 在 local memory 的标志
- shape -指向 input data 或 mask 的 shape 的指针
- data_dtype -input/output data 的数据类型
- mask_dtype -mask 的数据类型

注意事项

- mask_select 后的 filter_num 可由 tpu_gdma_get_filter_num() 得到。该函数没有输入参数，返回值的类型为DT_UINT32。

4.6.17 tpu_gdma_nonzero_L2S

将 local memory 的输入张量中不为 0 的元素的 index 输出到 global memory 中。

```
void tpu_gdma_nonzero_L2S(global_addr_t dst_addr, local_addr_t src_addr, const
                           dim4 *shape, data_type_t data_type, unsigned int
                           base_idx)
```

参数

- dst_addr -dst 在 global memory 中的地址
- src_addr -src 在 local memory 中的地址
- shape -指向输入张量的 shape 的指针
- data_type -输入张量的元素的数据类型
- base_idx -dst 的起始 index

注意事项

- dst 是 64-byte aligned layout, src 是 compact layout。
- dst index 的个数可由 tpu_gdma_get_filter_num() 得到。该函数没有输入参数，返回值的类型为DT_UINT32。

4.6.18 tpu_gdma_nonzero_S2S

将 global memory 的输入张量中不为 0 的元素的 index 输出到 global memory 中。

```
void tpu_gdma_nonzero_L2S(global_addr_t dst_addr, global_addr_t src_addr, const
    dim4 *shape, data_type_t data_type, unsigned int
    base_idx)
```

参数

- `dst_addr` -dst 在 global memory 中的地址
- `src_addr` -src 在 global memory 中的地址
- `shape` -指向输入张量的 shape 的指针
- `data_type` -输入张量的元素的数据类型
- `base_idx` -dst 的起始 index

注意事项

- `dst` 是 compact layout, `src` 是 compact layout。
- `dst` index 的个数可由 `tpu_gdma_get_filter_num()` 得到。该函数没有输入参数，返回值的类型为 `DT_UINT32`。

4.6.19 tpu_gdma_compact_S2L

张量的元素从 system memory 拷贝到 local memory。

```
void tpu_gdma_compact_S2L(local_addr_t dst_addr, system_addr_t src_addr, const
    dim4 *shape, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(n, c, h, w)$$

参数

- `dst_addr` -dst 在 local memory 中的地址
- `src_addr` -src 在 system memory 中的地址
- `shape` -指向 `dst` 和 `src` 的 shape 的指针
- `dtype` -dst 和 `src` 的元素的数据类型

注意事项

- `dst` 是 compact layout, `src` 是 continuous layout。

4.6.20 tpu_gdma_compact_L2S

张量的元素从 local memory 拷贝到 system memory。

```
void tpu_gdma_compact_L2S(system_addr_t dst_addr, local_addr_t src_addr, const
    dim4 *shape, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(n, c, h, w)$$

参数

- `dst_addr` -dst 在 system memory 中的地址
- `src_addr` -src 在 local memory 中的地址

- `shape` –指向 `dst` 和 `src` 的 `shape` 的指针
- `dtype` –`dst` 和 `src` 的元素的 datatype

注意事项

- `dst` 是 continuous layout, `src` 是 compact layout。

4.6.21 tpu_gdma_compact_nc_trans_S2L

张量的元素从 system memory 拷贝到 local memory, N 和 C 维度转置。

```
void tpu_gdma_compact_nc_trans_S2L(local_addr_t dst_addr, system_addr_t
                                   src_addr, const dim4 *dst_shape, data_type_t
                                   dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(c, n, h, w)$$

参数

- `dst_addr` –`dst` 在 local memory 中的地址
- `src_addr` –`src` 在 system memory 中的地址
- `dst_shape` –指向 `dst` 的 `shape` 的指针
- `dtype` –`dst` 和 `src` 的元素的 datatype

注意事项

- `src` 的 `shape` 是 [`dst_shape->c`, `dst_shape->n`, `dst_shape->h`, `dst_shape->w`]。
- `dst` 是 compact layout, `src` 是 continuous layout。

4.6.22 tpu_gdma_compact_nc_trans_L2S

张量的元素从 local memory 拷贝到 system memory, N 和 C 维度转置。

```
void tpu_gdma_compact_nc_trans_L2S(system_addr_t dst_addr, local_addr_t
                                   src_addr, const dim4 *dst_shape, data_type_t
                                   dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(c, n, h, w)$$

参数

- `dst_addr` –`dst` 在 system memory 中的地址
- `src_addr` –`src` 在 local memory 中的地址
- `dst_shape` –指向 `dst` 的 `shape` 的指针
- `dtype` –`dst` 和 `src` 的元素的 datatype

注意事项

- `src` 的 `shape` 是 [`dst_shape->c`, `dst_shape->n`, `dst_shape->h`, `dst_shape->w`]。
- `dst` 是 continuous layout, `src` 是 compact layout。

4.6.23 tpu_gdma_set_C_system

将 system memory 中的张量的元素置成常数。

```
void tpu_gdma_set_C_system(system_addr_t dst_addr, scalar_t C, const dim4 *shape,
                           const dim4 *dst_stride, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = C$$

参数

- dst_addr –dst 在 system memory 中的地址
- C –常数
- shape –指向 dst 的 shape 的指针
- dst_stride –指向 dst 的 stride 的指针
- dtype –dst 的元素和 C 的数据类型

注意事项

- 如果 dst_stride 是 NULL，则 dst 是 continuous layout，否则是 free layout。
- dst_stride->w 小于等于 128 / tpu_data_type_size(dtype)。

4.6.24 tpu_gdma_set_C_local

将 local memory 中的张量的元素置成常数。

```
void tpu_gdma_set_C_local(local_addr_t dst_addr, scalar_t C, const dim4 *shape,
                           const dim4 *dst_stride, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = C$$

参数

- dst_addr –dst 在 local memory 中的地址
- C –常数
- shape –指向 dst 的 shape 的指针
- dst_stride –指向 dst 的 stride 的指针
- dtype –dst 的元素和 C 的数据类型

注意事项

- 如果 dst_stride 是 NULL，则 dst 是 64-byte aligned layout，否则是 free layout。
- dst_stride->w 小于等于 128 / tpu_data_type_size(dtype)。

4.6.25 tpu_gdma_matrix_S2L

矩阵的元素从 system memory 拷贝到 local memory。

```
void tpu_gdma_matrix_S2L(local_addr_t dst_addr, system_addr_t src_addr, int rows,
                          int cols, int cols_per_channel, int row_stride, data_type_t dtype)
```

$$\text{dst}(x, y) = \text{src}(x, y)$$

参数

- `dst_addr` -dst 在 local memory 中的地址
- `src_addr` -src 在 system memory 中的地址
- `rows` -矩阵的行数
- `cols` -矩阵的列数
- `cols_per_channel` -dst 在每个 channel 的列数
- `row_stride` -src 的行 stride
- `dtype` -dst 和 src 的元素的数据类型

注意事项

- dst 是 matrix layout, src 的每一行的元素是连续存储, 通过 `row_stride` 换行。

4.6.26 tpu_gdma_matrix_L2S

矩阵的元素从 local memory 拷贝到 system memory。

```
void tpu_gdma_matrix_L2S(system_addr_t dst_addr, local_addr_t src_addr, int rows,
                        int cols, int cols_per_channel, int row_stride, data_type_t
                        dtype)
```

$$\text{dst}(x, y) = \text{src}(x, y)$$

参数

- `dst_addr` -dst 在 system memory 中的地址
- `src_addr` -src 在 local memory 中的地址
- `rows` -矩阵的行数
- `cols` -矩阵的列数
- `cols_per_channel` -src 在每个 channel 的列数
- `row_stride` -dst 的行 stride
- `dtype` -dst 和 src 的元素的数据类型

注意事项

- dst 的每一行的元素是连续存储, 通过 `row_stride` 换行, src 是 matrix layout。

4.6.27 tpu_gdma_matrix_trans_S2L

矩阵的元素从 system memory 转置拷贝到 local memory。

```
void tpu_gdma_matrix_trans_S2L(local_addr_t dst_addr, system_addr_t src_addr,
                                int src_rows, int src_cols, int dst_cols_per_channel,
                                int src_row_stride, data_type_t dtype)
```

$$\text{dst}(x, y) = \text{src}(y, x)$$

参数

- `dst_addr` -dst 在 local memory 中的地址
- `src_addr` -src 在 system memory 中的地址

- `src_rows` -src 的行数
- `src_cols` -src 的列数
- `dst_cols_per_channel` -dst 在每个 channel 的列数
- `src_row_stride` -src 的行 stride
- `dtype` -dst 和 src 的元素的数据类型

注意事项

- dst 是 matrix layout, src 的每一行的元素是连续存储, 通过 `src_row_stride` 换行。

4.6.28 tpu_gdma_matrix_trans_L2S

矩阵的元素从 local memory 转置拷贝到 system memory。

```
void tpu_gdma_matrix_trans_L2S(system_addr_t dst_addr, local_addr_t src_addr,
                               int src_rows, int src_cols, int src_cols_per_channel,
                               int dst_row_stride, data_type_t dtype)
```

$$\text{dst}(x, y) = \text{src}(y, x)$$

参数

- `dst_addr` -dst 在 system memory 中的地址
- `src_addr` -src 在 local memory 中的地址
- `src_rows` -src 的行数
- `src_cols` -src 的列数
- `src_cols_per_channel` -src 在每个 channel 的列数
- `dst_row_stride` -dst 的行 stride
- `dtype` -dst 和 src 的元素的数据类型

注意事项

- dst 的每一行的元素是连续存储, 通过 `dst_row_stride` 换行, src 是 matrix layout。

4.6.29 tpu_gdma_vector_S2L

向量的元素从 system memory 拷贝到 local memory。

```
void tpu_gdma_vector_S2L(local_addr_t dst_addr, system_addr_t src_addr, int len,
                          int len_per_channel, data_type_t dtype)
```

$$\text{dst}(x) = \text{src}(x)$$

参数

- `dst_addr` -dst 在 local memory 中的地址
- `src_addr` -src 在 system memory 中的地址
- `len` -向量的长度
- `len_per_channel` -dst 在每个 channel 的长度
- `dtype` -dst 和 src 的元素的数据类型

注意事项

- dst 是 vector layout, src 的元素是连续存储。

4.6.30 tpu_gdma_vector_L2S

向量的元素从 local memory 拷贝到 system memory。

```
void tpu_gdma_vector_L2S(system_addr_t dst_addr, local_addr_t src_addr, int len,
                        int len_per_channel, data_type_t dtype)
```

$$\text{dst}(x) = \text{src}(x)$$

参数

- `dst_addr` -dst 在 system memory 中的地址
- `src_addr` -src 在 local memory 中的地址
- `len` -向量的长度
- `len_per_channel` -src 在每个 channel 的长度
- `dtype` -dst 和 src 的元素的数据类型

注意事项

- `dst` 的元素是连续存储, `src` 是 vector layout。

4.6.31 tpu_gdma_channel_bcast_S2L

张量的元素从 system memory 拷贝到 local memory, channel 广播。

```
void tpu_gdma_channel_bcast_S2L(local_addr_t dst_addr, system_addr_t src_addr,
                                const dim4 *shape, const dim4 *dst_stride, const
                                dim4 *src_stride, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(n, 0, h, w)$$

参数

- `dst_addr` -dst 在 local memory 中的地址
- `src_addr` -src 在 system memory 中的地址
- `shape` -指向 dst 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针
- `src_stride` -指向 src 的 stride 的指针
- `dtype` -dst 和 src 的元素的数据类型

注意事项

- `src` 的 shape 是 [`shape->n`, 1, `shape->h`, `shape->w`]。
- 如果 `dst_stride` 是 NULL, 则 `dst` 是 64-byte aligned layout, 否则是 free layout。
- 如果 `src_stride` 是 NULL, 则 `src` 是 continuous layout, 否则是 free layout。
- 如果 `dst` 从 NPU `X` 开始, `X` 的取值范围是 `[0, NPU_NUM - 1]`, 且 `shape->c` 小于等于 `NPU_NUM - X`。
- `dst_stride->w` 和 `src_stride->w` 只能是 1。

4.6.32 tpu_gdma_channel_bcast_L2L

张量的元素从 local memory 拷贝到 local memory，channel 广播。

```
void tpu_gdma_channel_bcast_L2L(local_addr_t dst_addr, local_addr_t src_addr,
                                const dim4 *shape, const dim4 *dst_stride, const
                                dim4 *src_stride, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(n, 0, h, w)$$

参数

- `dst_addr` -dst 在 local memory 中的地址
- `src_addr` -src 在 local memory 中的地址
- `shape` -指向 dst 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针
- `src_stride` -指向 src 的 stride 的指针
- `dtype` -dst 和 src 的元素的数据类型

注意事项

- src 的 shape 是 [shape->n, 1, shape->h, shape->w]。
- 如果 `dst_stride` 是 NULL，则 dst 是 64-byte aligned layout，否则是 free layout。
- 如果 `src_stride` 是 NULL，则 src 是 64-byte aligned layout，否则是 free layout。
- 如果 dst 从 NPU X 开始，X 的取值范围是 [0, `NPU_NUM` - 1]，且 `shape->c` 小于等于 `NPU_NUM` - X。
- `dst_stride->w` 和 `src_stride->w` 只能是 1。

4.6.33 tpu_gdma_h_gather_S2L

通过 h 维度的索引取值得到输出张量，即 `output = param[index]`。

```
void tpu_gdma_h_gather_S2L(local_addr_t output_addr, system_addr_t param_addr,
                            addr_t index_addr, bool index_is_local, scalar_t C,
                            const dim4 *shape, int param_h, const dim4
                            *output_stride, const dim4 *param_stride, const dim4
                            *index_stride, data_type_t dtype)
```

$$\text{output}(0, c, h, w) = \begin{cases} \text{param}(0, c, \text{index}(0, c, h, 0), w) & \text{如果 index}(0, c, h, 0) \text{ 有效} \\ C & \text{其他情况} \end{cases}$$

参数

- `output_addr` -output 在 local memory 中的地址
- `param_addr` -param 在 system memory 中的地址
- `index_addr` -index 在 system memory 或 local memory 中的地址
- `index_is_local` -index 在 local memory 的标志
- `C` -常数
- `shape` -指向 output 的 shape 的指针
- `param_h` -param 的 h
- `output_stride` -指向 output 的 stride 的指针

- `param_stride` - 指向 `param` 的 `stride` 的指针
- `index_stride` - 指向 `index` 的 `stride` 的指针
- `dtype` - `output` 和 `param` 的元素的数据类型

注意事项

- 如果 `output_stride` 是 `NULL`, 则 `output` 是 64-byte aligned layout, 否则是 free layout。
- 如果 `param_stride` 是 `NULL`, 则 `param` 是 continuous layout, 否则是 free layout。
- 如果 `index_stride` 是 `NULL`, 则 `index` 是 64-byte aligned layout (`index_is_local` 是 true) 或 continuous layout (`index_is_local` 是 false), 否则是 free layout。
- 如果 `index_addr` 被 512 整除, 则性能更优。
- `shape->n` 只能是 1, `param` 的 `shape` 是 `[1, shape->c, param_h, shape->w]`, `index` 的 `shape` 是 `[1, shape->c, shape->h, 1]`。
- `output_stride->w`、`param_stride->w` 和 `index_stride->h` 只能是 1。
- `index` 的元素的数据类型是 `DT_UINT32`, 有效取值范围是 `[0, param_h - 1]`。

4.6.34 tpu_gdma_h_gather_L2S

通过 `h` 维度的索引取值得到输出张量, 即 `output = param[index]`。

```
void tpu_gdma_h_gather_L2S(system_addr_t output_addr, local_addr_t param_addr,
                           addr_t index_addr, bool index_is_local, scalar_t C,
                           const dim4 *shape, int param_h, const dim4
                           *output_stride, const dim4 *param_stride, const dim4
                           *index_stride, data_type_t dtype)
```

$$\text{output}(0, c, h, w) = \begin{cases} \text{param}(0, c, \text{index}(0, c, h, 0), w) & \text{如果 } \text{index}(0, c, h, 0) \text{ 有效} \\ C & \text{其他情况} \end{cases}$$

参数

- `output_addr` - `output` 在 system memory 中的地址
- `param_addr` - `param` 在 local memory 中的地址
- `index_addr` - `index` 在 system memory 或 local memory 中的地址
- `index_is_local` - `index` 在 local memory 的标志
- `C` - 常数
- `shape` - 指向 `output` 的 `shape` 的指针
- `param_h` - `param` 的 `h`
- `output_stride` - 指向 `output` 的 `stride` 的指针
- `param_stride` - 指向 `param` 的 `stride` 的指针
- `index_stride` - 指向 `index` 的 `stride` 的指针
- `dtype` - `output` 和 `param` 的元素的数据类型

注意事项

- 如果 `output_stride` 是 `NULL`, 则 `output` 是 continuous layout, 否则是 free layout。
- 如果 `param_stride` 是 `NULL`, 则 `param` 是 64-byte aligned layout, 否则是 free layout。
- 如果 `index_stride` 是 `NULL`, 则 `index` 是 64-byte aligned layout (`index_is_local` 是 true) 或 continuous layout (`index_is_local` 是 false), 否则是 free layout。

- 如果 `index_addr` 被 512 整除，则性能更优。
- `shape->n` 只能是 1，`param` 的 `shape` 是 `[1, shape->c, param_h, shape->w]`，`index` 的 `shape` 是 `[1, shape->c, shape->h, 1]`。
- `output_stride->w`、`param_stride->w` 和 `index_stride->h` 只能是 1。
- `index` 的元素的数据类型是 `DT_UINT32`，有效取值范围是 `[0, param_h - 1]`。

4.6.35 tpu_gdma_h_gather_L2L

通过 `h` 维度的索引取值得到输出张量，即 `output = param[index]`。

```
void tpu_gdma_h_gather_L2L(local_addr_t output_addr, local_addr_t param_addr,
                           addr_t index_addr, bool index_is_local, scalar_t C,
                           const dim4 *shape, int param_h, const dim4
                           *output_stride, const dim4 *param_stride, const dim4
                           *index_stride, data_type_t dtype)
```

$$\text{output}(0, c, h, w) = \begin{cases} \text{param}(0, c, \text{index}(0, c, h, 0), w) & \text{如果 } \text{index}(0, c, h, 0) \text{ 有效} \\ C & \text{其他情况} \end{cases}$$

参数

- `output_addr` - `output` 在 local memory 中的地址
- `param_addr` - `param` 在 local memory 中的地址
- `index_addr` - `index` 在 system memory 或 local memory 中的地址
- `index_is_local` - `index` 在 local memory 的标志
- `C` - 常数
- `shape` - 指向 `output` 的 `shape` 的指针
- `param_h` - `param` 的 `h`
- `output_stride` - 指向 `output` 的 `stride` 的指针
- `param_stride` - 指向 `param` 的 `stride` 的指针
- `index_stride` - 指向 `index` 的 `stride` 的指针
- `dtype` - `output` 和 `param` 的元素的数据类型

注意事项

- 如果 `output_stride` 是 `NULL`，则 `output` 是 64-byte aligned layout，否则是 free layout。
- 如果 `param_stride` 是 `NULL`，则 `param` 是 64-byte aligned layout，否则是 free layout。
- 如果 `index_stride` 是 `NULL`，则 `index` 是 64-byte aligned layout (`index_is_local` 是 true) 或 continuous layout (`index_is_local` 是 false)，否则是 free layout。
- 如果 `index_addr` 被 512 整除，则性能更优。
- `shape->n` 只能是 1，`param` 的 `shape` 是 `[1, shape->c, param_h, shape->w]`，`index` 的 `shape` 是 `[1, shape->c, shape->h, 1]`。
- `output_stride->w`、`param_stride->w` 和 `index_stride->h` 只能是 1。
- `index` 的元素的数据类型是 `DT_UINT32`，有效取值范围是 `[0, param_h - 1]`。

4.6.36 tpu_gdma_h_gather_S2S

通过 h 维度的索引取值得到输出张量，即 $\text{output} = \text{param}[\text{index}]$ 。

```
void tpu_gdma_h_gather_S2S(system_addr_t output_addr, system_addr_t
                           param_addr, addr_t index_addr, bool index_is_local,
                           scalar_t C, const dim4 *shape, int param_h, const dim4
                           *output_stride, const dim4 *param_stride, const dim4
                           *index_stride, data_type_t dtype)
```

$$\text{output}(0, c, h, w) = \begin{cases} \text{param}(0, c, \text{index}(0, c, h, 0), w) & \text{如果 } \text{index}(0, c, h, 0) \text{ 有效} \\ C & \text{其他情况} \end{cases}$$

参数

- output_addr - output 在 system memory 中的地址
- param_addr - param 在 system memory 中的地址
- index_addr - index 在 system memory 或 local memory 中的地址
- index_is_local - index 在 local memory 的标志
- C - 常数
- shape - 指向 output 的 shape 的指针
- param_h - param 的 h
- output_stride - 指向 output 的 stride 的指针
- param_stride - 指向 param 的 stride 的指针
- index_stride - 指向 index 的 stride 的指针
- dtype - output 和 param 的元素的数据类型

注意事项

- 如果 output_stride 是 NULL，则 output 是 continuous layout，否则是 free layout。
- 如果 param_stride 是 NULL，则 param 是 continuous layout，否则是 free layout。
- 如果 index_stride 是 NULL，则 index 是 64-byte aligned layout (index_is_local 是 true) 或 continuous layout (index_is_local 是 false)，否则是 free layout。
- 如果 index_addr 被 512 整除，则性能更优。
- shape->n 只能是 1，param 的 shape 是 [1, shape->c, param_h, shape->w]，index 的 shape 是 [1, shape->c, shape->h, 1]。
- output_stride->w、param_stride->w 和 index_stride->h 只能是 1。
- index 的元素的数据类型是 DT_UINT32，有效取值范围是 [0, param_h - 1]。

4.6.37 tpu_gdma_h_scatter_S2L

通过 h 维度的索引改变输出张量的对应元素，即 $\text{output}[\text{index}] = \text{param}$ 。

```
void tpu_gdma_h_scatter_S2L(local_addr_t output_addr, system_addr_t
                             param_addr, addr_t index_addr, bool index_is_local,
                             const dim4 *shape, int param_h, const dim4
                             *output_stride, const dim4 *param_stride, const dim4
                             *index_stride, data_type_t dtype)
```

$$\text{output}(0, c, \text{index}(0, c, h, 0), w) = \text{param}(0, c, h, w)$$

参数

- `output_addr` -output 在 local memory 中的地址
- `param_addr` -param 在 system memory 中的地址
- `index_addr` -index 在 system memory 或 local memory 中的地址
- `index_is_local` -index 在 local memory 的标志
- `shape` -指向 output 的 shape 的指针
- `param_h` -param 的 h
- `output_stride` -指向 output 的 stride 的指针
- `param_stride` -指向 param 的 stride 的指针
- `index_stride` -指向 index 的 stride 的指针
- `dtype` -output 和 param 的元素的数据类型

注意事项

- 如果 `output_stride` 是 NULL, 则 output 是 64-byte aligned layout, 否则是 free layout。
- 如果 `param_stride` 是 NULL, 则 param 是 continuous layout, 否则是 free layout。
- 如果 `index_stride` 是 NULL, 则 index 是 64-byte aligned layout (`index_is_local` 是 true) 或 continuous layout (`index_is_local` 是 false), 否则是 free layout。
- 如果 `index_addr` 被 512 整除, 则性能更优。
- `shape->n` 只能是 1, param 的 shape 是 `[1, shape->c, param_h, shape->w]`, index 的 shape 是 `[1, shape->c, param_h, 1]`。
- `output_stride->w`、`param_stride->w` 和 `index_stride->h` 只能是 1。
- index 的元素的数据类型是 `DT_UINT32`, 取值范围是 `[0, shape->n - 1]`。

4.6.38 tpu_gdma_h_scatter_L2S

通过 h 维度的索引改变输出张量的对应元素, 即 `output[index] = param`。

```
void tpu_gdma_h_scatter_L2S(system_addr_t output_addr, local_addr_t
                             param_addr, addr_t index_addr, bool index_is_local,
                             const dim4 *shape, int param_h, const dim4
                             *output_stride, const dim4 *param_stride, const dim4
                             *index_stride, data_type_t dtype)
```

$$\text{output}(0, c, \text{index}(0, c, h, 0), w) = \text{param}(0, c, h, w)$$
参数

- `output_addr` -output 在 system memory 中的地址
- `param_addr` -param 在 local memory 中的地址
- `index_addr` -index 在 system memory 或 local memory 中的地址
- `index_is_local` -index 在 local memory 的标志
- `shape` -指向 output 的 shape 的指针
- `param_h` -param 的 h
- `output_stride` -指向 output 的 stride 的指针
- `param_stride` -指向 param 的 stride 的指针

- `index_stride` - 指向 `index` 的 `stride` 的指针
- `dtype` - `output` 和 `param` 的元素的数据类型

注意事项

- 如果 `output_stride` 是 `NULL`, 则 `output` 是 `continuous layout`, 否则是 `free layout`。
- 如果 `param_stride` 是 `NULL`, 则 `param` 是 `64-byte aligned layout`, 否则是 `free layout`。
- 如果 `index_stride` 是 `NULL`, 则 `index` 是 `64-byte aligned layout` (`index_is_local` 是 `true`) 或 `continuous layout` (`index_is_local` 是 `false`), 否则是 `free layout`。
- 如果 `index_addr` 被 512 整除, 则性能更优。
- `shape->n` 只能是 1, `param` 的 `shape` 是 `[1, shape->c, param_h, shape->w]`, `index` 的 `shape` 是 `[1, shape->c, param_h, 1]`。
- `output_stride->w`, `param_stride->w` 和 `index_stride->h` 只能是 1。
- `index` 的元素的数据类型是 `DT_UINT32`, 取值范围是 `[0, shape->h - 1]`。

4.6.39 tpu_gdma_h_scatter_L2L

通过 `h` 维度的索引改变输出张量的对应元素, 即 `output[index] = param`。

```
void tpu_gdma_h_scatter_L2L(local_addr_t output_addr, local_addr_t param_addr,
                           addr_t index_addr, bool index_is_local, const dim4
                           *shape, int param_h, const dim4 *output_stride, const
                           dim4 *param_stride, const dim4 *index_stride,
                           data_type_t dtype)
```

$$\text{output}(0, c, \text{index}(0, c, h, 0), w) = \text{param}(0, c, h, w)$$

参数

- `output_addr` - `output` 在 `local memory` 中的地址
- `param_addr` - `param` 在 `local memory` 中的地址
- `index_addr` - `index` 在 `system memory` 或 `local memory` 中的地址
- `index_is_local` - `index` 在 `local memory` 的标志
- `shape` - 指向 `output` 的 `shape` 的指针
- `param_h` - `param` 的 `h`
- `output_stride` - 指向 `output` 的 `stride` 的指针
- `param_stride` - 指向 `param` 的 `stride` 的指针
- `index_stride` - 指向 `index` 的 `stride` 的指针
- `dtype` - `output` 和 `param` 的元素的数据类型

注意事项

- 如果 `output_stride` 是 `NULL`, 则 `output` 是 `64-byte aligned layout`, 否则是 `free layout`。
- 如果 `param_stride` 是 `NULL`, 则 `param` 是 `64-byte aligned layout`, 否则是 `free layout`。
- 如果 `index_stride` 是 `NULL`, 则 `index` 是 `64-byte aligned layout` (`index_is_local` 是 `true`) 或 `continuous layout` (`index_is_local` 是 `false`), 否则是 `free layout`。
- 如果 `index_addr` 被 512 整除, 则性能更优。
- `shape->n` 只能是 1, `param` 的 `shape` 是 `[1, shape->c, param_h, shape->w]`, `index` 的 `shape` 是 `[1, shape->c, param_h, 1]`。

- `output_stride->w`、`param_stride->w` 和 `index_stride->h` 只能是 1。
- `index` 的元素的数据类型是 `DT_UINT32`，取值范围是 `[0, shape->h - 1]`。

4.6.40 tpu_gdma_h_scatter_S2S

通过 `h` 维度的索引改变输出张量的对应元素，即 `output[index] = param`。

```
void tpu_gdma_h_scatter_S2S(system_addr_t output_addr, system_addr_t
                             param_addr, addr_t index_addr, bool index_is_local,
                             const dim4 *shape, int param_h, const dim4
                             *output_stride, const dim4 *param_stride, const dim4
                             *index_stride, data_type_t dtype)
```

`output(0, c, index(0, c, h, 0), w) = param(0, c, h, w)`

参数

- `output_addr` - output 在 system memory 中的地址
- `param_addr` - param 在 system memory 中的地址
- `index_addr` - index 在 system memory 或 local memory 中的地址
- `index_is_local` - index 在 local memory 的标志
- `shape` - 指向 output 的 shape 的指针
- `param_h` - param 的 `h`
- `output_stride` - 指向 output 的 stride 的指针
- `param_stride` - 指向 param 的 stride 的指针
- `index_stride` - 指向 index 的 stride 的指针
- `dtype` - output 和 param 的元素的数据类型

注意事项

- 如果 `output_stride` 是 NULL，则 output 是 continuous layout，否则是 free layout。
- 如果 `param_stride` 是 NULL，则 param 是 continuous layout，否则是 free layout。
- 如果 `index_stride` 是 NULL，则 index 是 64-byte aligned layout (`index_is_local` 是 true) 或 continuous layout (`index_is_local` 是 false)，否则是 free layout。
- 如果 `index_addr` 被 512 整除，则性能更优。
- `shape->n` 只能是 1，param 的 shape 是 `[1, shape->c, param_h, shape->w]`，index 的 shape 是 `[1, shape->c, param_h, 1]`。
- `output_stride->w`、`param_stride->w` 和 `index_stride->h` 只能是 1。
- `index` 的元素的数据类型是 `DT_UINT32`，取值范围是 `[0, shape->h - 1]`。

4.6.41 tpu_gdma_system_cpy

system memory 拷贝数据

```
void tpu_gdma_system_cpy(system_addr_t dst_addr, system_addr_t src_addr,
                          unsigned int count, data_type_t dtype)
```

参数

- `dst_addr` -dst 在 system memory 中的地址
- `src_addr` -src 在 system memory 中的地址
- `count` -数据的长度
- `dtype` -dst 和 src 的元素的数据类型

注意事项

- `dst_addr` 和 `src_addr` 都被元素的数据类型的位宽整除。

4.6.42 tpu_gdma_reverse_S2S

将 System Memory 中的数据，按照指定维度翻转顺序到另一块 System Memory 空间中。目前支持 N、C、H 维

```
void tpu_gdma_reverse_S2S(system_addr_t dst_addr, system_addr_t src_addr, dim4
                           *shape, dim4 *dst_stride, dim4 *src_stride, int
                           reverse_axis, data_type_t dtype);
```

参数

- `dst_addr` -dst 在 system memory 中的地址
- `src_addr` -src 在 system memory 中的地址
- `shape` -指向 dst 和 src 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针
- `src_stride` -指向 src 的 stride 的指针
- `reverse_axis` -指定在哪一维翻转，支持 0-N 维，1-C 维，2-H 维
- `dtype` -dst 和 src 的元素的数据类型

注意事项

- BM1684X 设备不支持该函数
- `dst_stride` 和 `src_stride` 可以为 NULL，用默认连续排列
- 如果提供 `src_stride`，必须保证 `src_stride->w` 为 1
- 如果提供 `dst_stride`，必须保证 `dst_stride->w` 为 1

4.6.43 tpu_gdma_reverse_S2L

将 System Memory 中的数据，按照指定维度翻转顺序到 Local Memory 空间中。目前支持 N、C、H 维

```
void tpu_gdma_reverse_S2L(local_addr_t dst_addr, system_addr_t src_addr, dim4
                          *shape, dim4 *dst_stride, dim4 *src_stride, int
                          reverse_axis, data_type_t dtype);
```

参数

- dst_addr -dst 在 local memory 中的地址
- src_addr -src 在 system memory 中的地址
- shape -指向 dst 和 src 的 shape 的指针
- dst_stride -指向 dst 的 stride 的指针
- src_stride -指向 src 的 stride 的指针
- reverse_axis -指定在哪一维翻转，支持：0-N 维，1-C 维，2-H 维
- dtype -dst 和 src 的元素的数据类型

注意事项

- BM1684X 设备不支持该函数
- dst_stride 和 src_stride 可以为 NULL，用默认连续排列
- 如果提供 src_stride，必须保证 src_stride->w 为 1
- 如果提供 dst_stride，必须保证 dst_stride->w 为 1

4.6.44 tpu_gdma_reverse_L2S

将 Local Memory 中的数据，按照指定维度翻转顺序到 System Memory 空间中。目前仅支持 C 维

```
void tpu_gdma_reverse_L2S(system_addr_t dst_addr, local_addr_t src_addr, dim4
                          *shape, dim4 *dst_stride, dim4 *src_stride, int
                          reverse_axis, data_type_t dtype);
```

参数

- dst_addr -dst 在 system memory 中的地址
- src_addr -src 在 local memory 中的地址
- shape -指向 dst 和 src 的 shape 的指针
- dst_stride -指向 dst 的 stride 的指针
- src_stride -指向 src 的 stride 的指针
- reverse_axis -指定在哪一维翻转，支持 1-C 维
- dtype -dst 和 src 的元素的数据类型

注意事项

- BM1684X 设备不支持该函数
- dst_stride 和 src_stride 可以为 NULL，用默认连续排列
- 如果提供 src_stride，必须保证 src_stride->w 为 1
- 如果提供 dst_stride，必须保证 dst_stride->w 为 1

4.6.45 tpu_gdma_reverse_L2L

将 Local Memory 中的数据，按照指定维度翻转顺序到另一 Local Memory 空间中。目前仅支持 C 维

```
void tpu_gdma_reverse_L2L(local_addr_t dst_addr, local_addr_t src_addr, dim4
                          *shape, dim4 *src_stride, dim4 *dst_stride, int
                          reverse_axis, data_type_t dtype);
```

参数

- `dst_addr` -dst 在 local memory 中的地址
- `src_addr` -src 在 local memory 中的地址
- `shape` -指向 dst 和 src 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针
- `src_stride` -指向 src 的 stride 的指针
- `reverse_axis` -指定在哪一维翻转，支持 1-C 维
- `dtype` -dst 和 src 的元素的数据类型

注意事项

- BM1684X 设备不支持该函数
- `dst_stride` 和 `src_stride` 可以为 NULL，用默认连续排列
- 如果提供 `src_stride`，必须保证 `src_stride->w` 为 1
- 如果提供 `dst_stride`，必须保证 `dst_stride->w` 为 1

4.6.46 tpu_gdma_compress_normal_L2S

将 Local Memory 空间中的数据压缩存储到 System Memory，该压缩算法不支持解压时随机访问，适用于加载权重或者部分不要切片访问 tensor 的场景

```
void tpu_gdma_compress_normal_L2S(global_addr_t dst_addr,
local_addr_t src_addr, dim4 *shape, dim4 *src_stride data_type_t dtype,
unsigned char bias0, unsigned char bias1, bool zero_guard);
```

参数

- `dst_addr` -dst 在 system memory 中的地址
- `src_addr` -src 在 local memory 中的地址
- `shape` -指向 dst 和 src 的 shape 的指针
- `src_stride` -指向 src 的 stride 的指针
- `dtype` -dst 和 src 的元素的数据类型
- `bias0` -为压缩参数
- `bias1` -为压缩参数
- `zero_guard` -是否把 denormal 看作 0（仅对 fp16 有效，bf16 默认 true，其他数据类型默认 false）

注意事项

- BM1684X 设备不支持该函数
- `src_stride` 可以为 NULL，用默认连续排列

- 如果提供 `src_stride`, 必须保证 `src_stride->w` 为 1
- 被压缩值变换的偏置值 (`bias0`、`bias1` 均为 8bit 无符整型), 选择合适的参数可以提升压缩率

4.6.47 tpu_gdma_decompress_normal_S2L

将用 `tpu_gdma_compress_normal_L2S` 生成的在 System Memory 空间中的压缩数据解压到 Local Memory

```
void tpu_gdma_decompress_normal_S2L(local_addr_t dst_addr,
                                     global_addr_t src_addr, dim4
                                     *shape, dim4 *dst_stride,
                                     data_type_t dtype, unsigned char
                                     bias0, unsigned char bias1, bool
                                     zero_guard);
```

参数

- `dst_addr` -dst 在 local memory 中的地址
- `src_addr` -src 在 system memory 中的地址
- `shape` -指向 dst 和 src 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针
- `dtype` -dst 和 src 的元素的数据类型
- `zero_guard` -是否把 denormal 看作 0 (仅对 fp16 有效, bf16 默认 true, 其他数据类型默认 false)

注意事项

- BM1684X 设备不支持该函数
- `src_stride` 可以为 NULL, 用默认连续排列
- 如果提供 `src_stride`, 必须保证 `src_stride->w` 为 1
- 被压缩值变换的偏置值 (`bias0`、`bias1` 均为 8bit 无符整型), 必须和压缩时的参数一致

`tpu_gdma_compress_normal_max_bytes`

计算压缩后的数据最大字节数, 用于 system memory 的空间分配

4.6.48 tpu_gdma_compress_RACU_L2S

将 Local Memory 空间中的数据压缩存储到 System Memory, 该压缩算法支持解压时随机访问, 适用于重复访问 tensor 切片的场景. 算法支持最小按照 RACU (Random Access Compression Unit) 粒度进行压缩数据的访问. 压缩后会输出 RACU 和 Meta 两块数据

```
void tpu_gdma_compress_RACU_L2S(global_addr_t dst_racu_addr, global_addr_t
                                  dst_meta_addr, local_addr_t src_addr, dim4
                                  *shape, dim4 *dst_racu_stride, dim4
                                  *dst_meta_stride, dim4 *src_stride,
                                  data_type_t dtype, unsigned char bias0,
                                  unsigned char bias1, bool zero_guard);
```

参数

- `dst_racu_addr` -dst 在 system memory 中 racu 数据地址
- `dst_meta_addr` -dst 在 system memory 中 meta 数据 j 地址

- `src_addr` -src 在 local memory 中的地址
- `shape` -指向 dst 和 src 的 shape 的指针
- `dst_racu_stride` -输出参数，压缩数据并会计算 racu 数据的 stride，供随机访问解压缩使用
- `dst_meta_stride` -输出参数，压缩数据并会计算 meta 数据的 stride，供随机访问解压缩使用
- `src_stride` -指向 src 的 stride 的指针
- `dtype` -dst 和 src 的元素的数据类型
- `bias0` -为压缩参数
- `bias1` -为压缩参数
- `zero_guard` -是否把 denormal 看作 0（仅对 fp16 有效，bf16 默认 true，其他数据类型默认 false）

注意事项

- BM1684X 设备不支持该函数
- `src_stride` 可以为 NULL，用默认连续排列
- 如果提供 `src_stride`，必须保证 `src_stride->w` 为 1
- 被压缩值变换的偏置值 (`bias0`、`bias1` 均为 8bit 无符整型)，选择合适的参数可以提升压缩率

4.6.49 tpu_gdma_decompress_RACU_S2L

将使用 `tpu_gdma_compress_RACU_L2S` 操作压缩在 System Memory 空间中的数据解压到 Local Memory

```
void tpu_gdma_compress_RACU_L2S(local_addr_t dst_addr, global_addr_t
                                src_racu_addr, global_addr_t
                                src_meta_addr, dim4 *shape, dim4
                                *dst_stride, dim4 *src_racu_stride,
                                dim4 *src_meta_stride, data_type_t
                                dtype, unsigned char bias0, unsigned
                                char bias1, bool zero_guard)
```

参数

- `dst_addr` -dst 在 local memory 中的地址
- `src_racu_addr` -src 在 system memory 中 racu 数据地址
- `src_meta_addr` -src 在 system memory 中 meta 数据 j 地址
- `shape` -指向 dst 和 src 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针
- `src_racu_stride` -指向 src racu 数据的 stride 的指针，只有 n, c 值有效，h、w 值会被忽略
- `src_meta_stride` -指向 src meta 数据的 stride 的指针，只有 n, c 值有效，h、w 值会被忽略
- `dtype` -dst 和 src 的元素的数据类型
- `bias0` -为压缩参数，必须和压缩时的参数一致
- `bias1` -为压缩参数，必须和压缩时的参数一致
- `zero_guard` -是否把 denormal 看作 0（仅对 fp16 有效，bf16 默认 true，其他数据类型默认 false）

注意事项

- BM1684X 设备不支持该函数
- `dst_stride` 可以为 NULL，用默认连续排列

- 如果提供 src_stride, 必须保证 dst_stride->w 为 1

```
tpu_gdma_compress_RACU_max_racu_bytes
```

计算 RACU 算法中压缩后的 racu 数据最大字节数, 用于 system memory 的空间分配

```
int tpu_gdma_compress_RACU_max_racu_bytes(const dim4 *shape,
                                           data_type_t dtype);
```

参数

- shape - 待压缩数据的 shape 指针
- dtype - 待压缩数据的数据类型

```
tpu_gdma_compress_RACU_max_meta_bytes
```

计算 RACU 算法中压缩后的 meta 数据最大字节数, 用于 system memory 的空间分配

```
int tpu_gdma_compress_RACU_max_meta_bytes(const dim4 *shape, data_type_t
                                           dtype);
```

参数

- shape - 待压缩数据的 shape 指针
- dtype - 待压缩数据的数据类型

4.7 基础数据操作

4.7.1 tpu_bdc_cpy

拷贝张量的元素。

```
void tpu_bdc_cpy(local_addr_t dst_addr, local_addr_t src_addr, const dim4 *shape,
                 const dim4 *dst_stride, const dim4 *src_stride, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(n, c, h, w)$$

参数

- dst_addr - dst 的地址
- src_addr - src 的地址
- shape - 指向 dst 和 src 的 shape 的指针
- dst_stride - 指向 dst 的 stride 的指针
- src_stride - 指向 src 的 stride 的指针
- dtype - dst 和 src 的元素的数据类型

注意事项

- dst 和 src 从同一个 NPU 开始。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL, 则此张量是 64-byte aligned layout, 否则是 free layout。

4.7.2 tpu_bdc_cpy_cross_npu

跨 NPU 拷贝张量的元素。

```
void tpu_bdc_cpy_cross_npu(local_addr_t dst_addr, local_addr_t src_addr, const
                           dim4 *shape, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(n, c, h, w)$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `shape` -指向 dst 和 src 的 shape 的指针
- `dtype` -dst 和 src 的元素的数据类型

注意事项

- `dst` 和 `src` 可以不从同一个 NPU 开始，都是 64-byte aligned layout。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。

4.7.3 tpu_bdc_npu_bcast

广播张量的元素到其他 NPU。

```
void tpu_bdc_npu_bcast(local_addr_t dst_addr, local_addr_t src_addr, const dim4
                       *shape, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(n, 0, h, w)$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `shape` -指向 dst 的 shape 的指针
- `dtype` -dst 和 src 的元素的数据类型

注意事项

- `dst` 和 `src` 可以不从同一个 NPU 开始，都是 64-byte aligned layout。
- `src` 的 shape 是 [`shape->n`, 1, `shape->h`, `shape->w`]。
- `shape->n`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。
- 如果 `dst` 从 NPU `X` 开始，`X` 的取值范围是 [0, `NPU_NUM` - 1]，则 `shape->c` 小于等于 `NPU_NUM` - `X`。

4.7.4 tpu_bdc_set_C

将张量的元素置成常数。

```
void tpu_bdc_set_C(local_addr_t dst_addr, scalar_t C, const dim4 *shape, const dim4
                  *dst_stride, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = C$$

参数

- dst_addr -dst 的地址
- C -常数
- shape -指向 dst 的 shape 的指针
- dst_stride -指向 dst 的 stride 的指针
- dtype -dst 的元素和 C 的数据类型

注意事项

- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。

4.7.5 tpu_bdc_cw_trans

张量的 C 与 W 两个维度转置，推荐在 tensor C>W 时使用。

```
void tpu_bdc_cw_trans(local_addr_t dst_addr, local_addr_t src_addr, const dim4
                    *shape, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(n, w, h, c)$$

参数

- dst_addr -dst 的地址
- src_addr -src 的地址
- shape -指向 dst 的 shape 的指针
- dtype -dst 和 src 的元素的数据类型

注意事项

- src 从 NPU 0 开始，64-byte aligned layout，dst 是 line 64-byte aligned layout。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。

4.7.6 tpu_bdc_wc_trans

张量的 W 与 C 两个维度转置，推荐在 tensor W>C 时使用。

```
void tpu_bdc_wc_trans(local_addr_t dst_addr, local_addr_t src_addr, const dim4
                    *shape, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(n, w, h, c)$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `shape` -指向 dst 的 shape 的指针
- `dtype` -dst 和 src 的元素的数据类型

注意事项

- dst 从 NPU 0 开始, dst 和 src 都是 64-byte aligned layout。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。

4.8 数据类型转换与舍入操作

4.8.1 `tpu_bdc_cast`

转换张量的元素的数据类型, 对结果做 saturation。

```
void tpu_bdc_cast(local_addr_t dst_addr, local_addr_t src_addr, const dim4 *shape,
                  const dim4 *dst_stride, const dim4 *src_stride, data_type_t
                  dst_dtype, data_type_t src_dtype, rounding_mode_t mode)
```

$$\text{dst}(n, c, h, w) = \text{CAST}(\text{src}(n, c, h, w))$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `shape` -指向 dst 和 src 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针
- `src_stride` -指向 src 的 stride 的指针
- `dst_dtype` -dst 的元素的数据类型
- `src_dtype` -src 的元素的数据类型
- `mode` -舍入模式

注意事项

- dst 和 src 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL, 则此张量是 64-byte aligned layout, 否则是 free layout。
- 不支持 `DT_FP16` 和 `DT_BFP16` 的数据类型互转, 如果 `dst_dtype` 与 `src_dtype` 相同, 则 dst 与 src 的元素完全相同。
- `mode` 只有在浮点数据类型参与的转换时使用, 有效取值是 `RM_HALF_TO_EVEN`、`RM_HALF_AWAY_FROM_ZERO`、`RM_TOWARDS_ZERO`、`RM_DOWN` 和 `RM_UP`。
- 下表中列出数据类型转换时 `mode` 是否有效, * 代表有效, 空白代表无效。

src - dst	FP32	FP16	BFP16	INT32	UINT32	INT16	UINT16	INT8	UINT8
FP32		*	*	*	*	*	*	*	*
FP16				*	*	*	*	*	*
BFP16				*	*	*	*	*	*
INT32	*	*	*						
UINT32	*	*	*						
INT16		*	*						
UINT16		*	*						
INT8									
UINT8									

4.8.2 tpu_bdc_fp_round

浮点类型的张量的元素舍入到附近的同类型的整数。

```
void tpu_bdc_fp_round(local_addr_t dst_addr, local_addr_t src_addr, const dim4
                      *shape, const dim4 *dst_stride, const dim4 *src_stride,
                      data_type_t dtype, rounding_mode_t mode)
```

$$\text{dst}(n, c, h, w) = \text{round}(\text{src}(n, c, h, w))$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `shape` -指向 dst 和 src 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针
- `src_stride` -指向 src 的 stride 的指针
- `dtype` -dst 和 src 的元素的数据类型
- `mode` -舍入模式

注意事项

- dst 和 src 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- `dtype` 的有效取值是 `DT_FP32`、`DT_FP16` 和 `DT_BFP16`。
- `mode` 的有效取值是 `RM_HALF_TO_EVEN`、`RM_HALF_AWAY_FROM_ZERO`、`RM_TOWARDS_ZERO`、`RM_DOWN` 和 `RM_UP`。

4.8.3 tpu_bdc_fp_floor

浮点类型的张量的元素向负无穷舍入到附近的同类型的整数。

```
void tpu_bdc_fp_floor(local_addr_t dst_addr, local_addr_t src_addr, const dim4
                      *shape, const dim4 *dst_stride, const dim4 *src_stride,
                      data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{floor}(\text{src}(n, c, h, w))$$

参数

- `dst_addr` -dst 的地址

- `src_addr` -src 的地址
- `shape` -指向 dst 和 src 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针
- `src_stride` -指向 src 的 stride 的指针
- `dtype` -dst 和 src 的元素的数据类型

注意事项

- dst 和 src 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- `dtype` 的有效取值是 `DT_FP32`、`DT_FP16` 和 `DT_BFP16`。

4.8.4 tpu_bdc_fp_ceil

浮点类型的张量的元素向正无穷舍入到附近的同类型的整数。

```
void tpu_bdc_fp_ceil(local_addr_t dst_addr, local_addr_t src_addr, const dim4
                    *shape, const dim4 *dst_stride, const dim4 *src_stride,
                    data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{ceil}(\text{src}(n, c, h, w))$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `shape` -指向 dst 和 src 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针
- `src_stride` -指向 src 的 stride 的指针
- `dtype` -dst 和 src 的元素的数据类型

注意事项

- dst 和 src 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- `dtype` 的有效取值是 `DT_FP32`、`DT_FP16` 和 `DT_BFP16`。

4.9 一元操作

4.9.1 tpu_bdc_abs

取张量的元素的绝对值。

```
void tpu_bdc_abs(local_addr_t dst_addr, local_addr_t src_addr, const dim4 *shape,
                 const dim4 *dst_stride, const dim4 *src_stride, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = |\text{src}(n, c, h, w)|$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `shape` -指向 dst 和 src 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针
- `src_stride` -指向 src 的 stride 的指针
- `dtype` -dst 和 src 的元素的数据类型

注意事项

- dst 和 src 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。

4.9.2 tpu_bdc_not

张量的元素按位取反。

```
void tpu_bdc_not(local_addr_t dst_addr, local_addr_t src_addr, const dim4 *shape,
                 const dim4 *dst_stride, const dim4 *src_stride, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{NOT}(\text{src}(n, c, h, w))$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `shape` -指向 dst 和 src 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针
- `src_stride` -指向 src 的 stride 的指针
- `dtype` -dst 和 src 的元素的数据类型

注意事项

- dst 和 src 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。

4.9.3 tpu_bdc_neg

张量的元素取相反数。

```
void tpu_bdc_neg(local_addr_t dst_addr, local_addr_t src_addr, const dim4 *shape,
                const dim4 *dst_stride, const dim4 *src_stride, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = -\text{src}(n, c, h, w)$$

参数

- dst_addr -dst 的地址
- src_addr -src 的地址
- shape -指向 dst 和 src 的 shape 的指针
- dst_stride -指向 dst 的 stride 的指针
- src_stride -指向 src 的 stride 的指针
- dtype -dst 和 src 的元素的数据类型

注意事项

- dst 和 src 从同一个 NPU 开始。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- 如果 dtype 是 DT_UINT32、DT_UINT16 或 DT_UINT8，则 dst 的元素的数据类型是 DT_INT32、DT_INT16 或 DT_INT8。

4.9.4 tpu_bdc_fp32_reciprocal

张量的元素取倒数。

```
void tpu_bdc_fp32_reciprocal(local_addr_t dst_addr, local_addr_t src_addr, const
                             dim4 *shape, const dim4 *dst_stride, const dim4
                             *src_stride)
```

$$\text{dst}(n, c, h, w) = \frac{1}{\text{src}(n, c, h, w)}$$

参数

- dst_addr -dst 的地址
- src_addr -src 的地址
- shape -指向 dst 和 src 的 shape 的指针
- dst_stride -指向 dst 的 stride 的指针
- src_stride -指向 src 的 stride 的指针

注意事项

- dst 和 src 从同一个 NPU 开始。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- 等价于 Newton 迭代次数是 3 的 tpu_bdc_fp32_tunable_reciprocal()。

4.9.5 tpu_bdc_fp32_tunable_reciprocal

张量的元素取倒数。

```
void tpu_bdc_fp32_tunable_reciprocal(local_addr_t dst_addr, local_addr_t src_addr,
                                     const dim4 *shape, const dim4 *dst_stride, const
                                     dim4 *src_stride, int num_iter)
```

$$\text{dst}(n, c, h, w) = \frac{1}{\text{src}(n, c, h, w)}$$

参数

- dst_addr -dst 的地址
- src_addr -src 的地址
- shape -指向 dst 和 src 的 shape 的指针
- dst_stride -指向 dst 的 stride 的指针
- src_stride -指向 src 的 stride 的指针
- num_iter -Newton 迭代次数

注意事项

- dst 和 src 从同一个 NPU 开始。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- num_iter 是性能参数，取值范围是 [1, 4]。

4.9.6 tpu_bdc_fp32_rsqr

张量的元素的算术平方根倒数。

```
void tpu_bdc_fp32_rsqr(local_addr_t dst_addr, local_addr_t src_addr, const dim4
                       *shape)
```

$$\text{dst}(n, c, h, w) = \frac{1}{\sqrt{\text{src}(n, c, h, w)}}$$

参数

- dst_addr -dst 的地址
- src_addr -src 的地址
- shape -指向 dst 和 src 的 shape 的指针

注意事项

- dst 和 src 从同一个 NPU 开始，都是 64-byte aligned layout。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- 等价于 Newton 迭代次数是 3 的 tpu_bdc_fp32_tunable_rsqr()。

4.9.7 tpu_bdc_fp32_tunable_rsqrt

张量的元素的算术平方根倒数。

```
void tpu_bdc_fp32_tunable_rsqrt(local_addr_t dst_addr, local_addr_t src_addr, const
                                dim4 *shape, int num_iter)
```

$$\text{dst}(n, c, h, w) = \frac{1}{\sqrt{\text{src}(n, c, h, w)}}$$

参数

- dst_addr -dst 的地址
- src_addr -src 的地址
- shape -指向 dst 和 src 的 shape 的指针
- num_iter -Newton 迭代次数

注意事项

- dst 和 src 从同一个 NPU 开始，都是 64-byte aligned layout。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- num_iter 是性能参数，取值范围是 [1, 4]。

4.9.8 tpu_bdc_fp32_sqrt

张量的元素的算术平方根。

```
void tpu_bdc_fp32_sqrt(local_addr_t dst_addr, local_addr_t src_addr, const dim4
                       *shape)
```

$$\text{dst}(n, c, h, w) = \sqrt{\text{src}(n, c, h, w)}$$

参数

- dst_addr -dst 的地址
- src_addr -src 的地址
- shape -指向 dst 和 src 的 shape 的指针

注意事项

- dst 和 src 从同一个 NPU 开始，都是 64-byte aligned layout，不允许 dst_addr 与 src_addr 相同。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- 等价于 Newton 迭代次数是 3 的 tpu_bdc_fp32_tunable_sqrt()。

4.9.9 tpu_bdc_fp32_tunable_sqrt

张量的元素的算术平方根。

```
void tpu_bdc_fp32_tunable_sqrt(local_addr_t dst_addr, local_addr_t src_addr, const
                                dim4 *shape, int num_iter)
```

$$\text{dst}(n, c, h, w) = \sqrt{\text{src}(n, c, h, w)}$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `shape` -指向 dst 和 src 的 shape 的指针
- `num_iter` -Newton 迭代次数

注意事项

- dst 和 src 从同一个 NPU 开始，都是 64-byte aligned layout，不允许 `dst_addr` 与 `src_addr` 相同。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。
- `num_iter` 是性能参数，取值范围是 [1, 4]。

4.9.10 tpu_bdc_fp32_exp

张量的元素为指数，自然常数 e 为底数的指数运算。

```
void tpu_bdc_fp32_exp(local_addr_t dst_addr, local_addr_t src_addr, local_addr_t
                     work0_addr, local_addr_t work1_addr, local_addr_t
                     coeff_addr, local_addr_t table_addr, const dim4 *shape)
```

$$\text{dst}(n, c, h, w) = e^{\text{src}(n, c, h, w)}$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `work0_addr` -work0 的地址
- `work1_addr` -work1 的地址
- `coeff_addr` -coeff 的地址
- `table_addr` -table 的地址
- `shape` -dst、src、work0 和 work1 的 shape 的指针

注意事项

- dst、src、work0 和 work1 从同一个 NPU 开始，都是 64-byte aligned layout。
- coeff 通过 `tpu_bdc_load_fp32_exp_coeff()` 加载，table 通过 `tpu_bdc_load_fp32_exp_table()` 加载。
- work0 和 work1 存放中间结果，不允许 `dst_addr`、`work0_addr` 和 `work1_addr` 任意两个相等，允许 `src_addr` 等于 `dst_addr`、`work0_addr` 或 `work1_addr`，此情况下 src 的数据会被覆盖，如果 dst、work1 和 table 两两之间没有 bank conflicting，则性能更优。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]，`shape->h * shape->w` 小于等于 65535。
- 如果 `src(n, c, h, w)` 小于 -103 或大于 88，则 $e^{\text{src}(n, c, h, w)}$ 分别是 e^{-103} 和 e^{88} 。

4.9.11 tpu_bdc_fp32_expm1

张量的元素为指数，自然常数 e 为底数的指数运算，结果再减 1。

```
void tpu_bdc_fp32_expm1(local_addr_t dst_addr, local_addr_t src_addr,
                        local_addr_t work0_addr, local_addr_t work1_addr,
                        local_addr_t coeff_addr, local_addr_t table_addr, const
                        dim4 *shape)
```

$$\text{dst}(n, c, h, w) = e^{\text{src}(n, c, h, w)} - 1$$

参数

- dst_addr -dst 的地址
- src_addr -src 的地址
- work0_addr -work0 的地址
- work1_addr -work1 的地址
- coeff_addr -coeff 的地址
- table_addr -table 的地址
- shape -dst、src、work0 和 work1 的 shape 的指针

注意事项

- dst、src、work0 和 work1 从同一个 NPU 开始，都是 64-byte aligned layout。
- coeff 通过 tpu_bdc_load_fp32_exp_coeff() 加载，table 通过 tpu_bdc_load_fp32_exp_table() 加载。
- work0 和 work1 存放中间结果，不允许 dst_addr、work0_addr 和 work1_addr 任意两个相等，允许 src_addr 等于 dst_addr、work0_addr 或 work1_addr，此情况下 src 的数据会被覆盖，如果 dst、work1 和 table 两两之间没有 bank conflicting，则性能更优。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]，shape->h * shape->w 小于等于 65535。
- 如果 src(n, c, h, w) 小于 -103 或大于 88，则 $e^{\text{src}(n, c, h, w)}$ 分别是 e^{-103} 和 e^{88} 。

4.9.12 tpu_bdc_fp32_log

以自然常数 e 为底数对张量的元素做对数运算。

```
void tpu_bdc_fp32_log(local_addr_t dst_addr, local_addr_t src_addr, local_addr_t
                      work_addr, local_addr_t coeff_addr, const dim4 *shape)
```

$$\text{dst}(n, c, h, w) = \log(\text{src}(n, c, h, w))$$

参数

- dst_addr -dst 的地址
- src_addr -src 的地址
- work_addr -work 的地址
- coeff_addr -coeff 的地址
- shape -dst、src 和 work 的 shape 的指针

注意事项

- dst、src 和 work 从同一个 NPU 开始，都是 64-byte aligned layout。

- coeff 通过 `tpu_bdc_load_fp32_log_coeff()` 加载。
- work 存放中间结果，不允许 `dst_addr` 等于 `work_addr`，允许 `src_addr` 等于 `dst_addr` 或 `work_addr`，此情况下 `src` 的数据会被覆盖。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 `[1, 65535]`。

4.9.13 tpu_bdc_fp32_log1p

以自然常数 e 为底数对张量的元素加 1 的结果做对数运算。

```
void tpu_bdc_fp32_log1p(local_addr_t dst_addr, local_addr_t src_addr, local_addr_t
                        work_addr, local_addr_t coeff_addr, const dim4 *shape)
```

$$\text{dst}(n, c, h, w) = \log(\text{src}(n, c, h, w) + 1)$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `work_addr` -work 的地址
- `coeff_addr` -coeff 的地址
- `shape` -dst、src 和 work 的 shape 的指针

注意事项

- dst、src 和 work 从同一个 NPU 开始，都是 64-byte aligned layout。
- coeff 通过 `tpu_bdc_load_fp32_log_coeff()` 加载。
- work 存放中间结果，不允许 `dst_addr` 等于 `work_addr`，允许 `src_addr` 等于 `dst_addr` 或 `work_addr`，此情况下 `src` 的数据会被覆盖。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 `[1, 65535]`。

4.9.14 tpu_bdc_fp32_logx

以 x 为底数对张量的元素做对数运算。

```
void tpu_bdc_fp32_logx(local_addr_t dst_addr, local_addr_t src_addr, local_addr_t
                       work_addr, local_addr_t coeff_addr, const dim4 *shape, float
                       x)
```

$$\text{dst}(n, c, h, w) = \log_x(\text{src}(n, c, h, w))$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `work_addr` -work 的地址
- `coeff_addr` -coeff 的地址
- `shape` -dst、src 和 work 的 shape 的指针
- `x` -底数

注意事项

- dst、src 和 work 从同一个 NPU 开始，都是 64-byte aligned layout。

- coeff 通过 `tpu_bdc_load_fp32_log_coeff()` 加载。
- work 存放中间结果，不允许 `dst_addr` 等于 `work_addr`，允许 `src_addr` 等于 `dst_addr` 或 `work_addr`，此情况下 `src` 的数据会被覆盖。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 `[1, 65535]`。
- `x` 大于 0。

4.9.15 tpu_bdc_sign

张量的元素做 sign 操作。

```
void tpu_bdc_sign(local_addr_t dst_addr, local_addr_t src_addr, const dim4 *shape,
                 data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \begin{cases} 1 & \text{如果 } \text{src}(n, c, h, w) > 0 \\ 0 & \text{如果 } \text{src}(n, c, h, w) = 0 \\ -1 & \text{如果 } \text{src}(n, c, h, w) < 0 \end{cases}$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `shape` -指向 dst 的 shape 的指针
- `dtype` -dst 和 src 的元素的数据类型

注意事项

- `dst` 和 `src` 从同一个 NPU 开始，都是 64-byte aligned layout。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 `[1, 65535]`。
- 如果 `dtype` 是无符号的，则允许 `dst_addr` 与 `src_addr` 相同。

4.9.16 tpu_bdc_fp32_sin

对张量的元素做 sin 运算。

```
void tpu_bdc_fp32_sin(local_addr_t dst_addr, local_addr_t src_addr, local_addr_t
                     work_addr, local_addr_t coeff_addr, const dim4 *shape)
```

$$\text{dst}(n, c, h, w) = \sin(\text{src}(n, c, h, w))$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `work_addr` -work 的地址
- `coeff_addr` -coeff 的地址
- `shape` -dst、src 和 work 的 shape 的指针

注意事项

- `dst`、`src` 和 `work` 从同一个 NPU 开始，都是 64-byte aligned layout。
- coeff 通过 `tpu_bdc_load_fp32_sin_coeff()` 加载。

- work 存放中间结果, 不允许 `dst_addr` 等于 `work_addr`, 允许 `src_addr` 等于 `dst_addr` 或 `work_addr`, 此情况下 `src` 的数据会被覆盖。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 `[1, 65535]`。

4.9.17 tpu_bdc_fp32_cos

对张量的元素做 `cos` 运算。

```
void tpu_bdc_fp32_cos(local_addr_t dst_addr, local_addr_t src_addr, local_addr_t
                     work_addr, local_addr_t coeff_addr, const dim4 *shape)
```

$$\text{dst}(n, c, h, w) = \sin(\text{src}(n, c, h, w))$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `work_addr` -work 的地址
- `coeff_addr` -coeff 的地址
- `shape` -dst、src 和 work 的 shape 的指针

注意事项

- dst、src 和 work 从同一个 NPU 开始, 都是 64-byte aligned layout。
- coeff 通过 `tpu_bdc_load_fp32_cos_coeff()` 加载。
- work 存放中间结果, 不允许 `dst_addr` 等于 `work_addr`, 允许 `src_addr` 等于 `dst_addr` 或 `work_addr`, 此情况下 `src` 的数据会被覆盖。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 `[1, 65535]`。

4.9.18 tpu_bdc_fp32_tan

对张量的元素做 `tan` 运算。

```
void tpu_bdc_fp32_tan(local_addr_t dst_addr, local_addr_t src_addr, local_addr_t
                     work_addr, local_addr_t coeff_addr, const dim4 *shape)
```

$$\text{dst}(n, c, h, w) = \tan(\text{src}(n, c, h, w))$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `work_addr` -work 的地址
- `coeff_addr` -coeff 的地址
- `shape` -dst、src 和 work 的 shape 的指针

注意事项

- dst、src 和 work 从同一个 NPU 开始, 都是 64-byte aligned layout。
- coeff 通过 `tpu_bdc_load_fp32_tan_coeff()` 加载。
- work 存放中间结果, 不允许 `dst_addr`、`src_addr` 和 `work_addr` 任意两个相等。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 `[1, 65535]`。

4.9.19 tpu_bdc_fp32_cot

对张量的元素做 cot 运算。

```
void tpu_bdc_fp32_cot(local_addr_t dst_addr, local_addr_t src_addr, local_addr_t
                      work_addr, local_addr_t coeff_addr, const dim4 *shape)
```

$$\text{dst}(n, c, h, w) = \text{cot}(\text{src}(n, c, h, w))$$

参数

- dst_addr -dst 的地址
- src_addr -src 的地址
- work_addr -work 的地址
- coeff_addr -coeff 的地址
- shape -dst、src 和 work 的 shape 的指针

注意事项

- dst、src 和 work 从同一个 NPU 开始，都是 64-byte aligned layout。
- coeff 通过 `tpu_bdc_load_fp32_tan_coeff()` 加载。
- work 存放中间结果，不允许 dst_addr、src_addr 和 work_addr 任意两个相等。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。

4.9.20 tpu_bdc_fp32_arcsin

对张量的元素做 arcsin 运算。

```
void tpu_bdc_fp32_arcsin(local_addr_t dst_addr, local_addr_t src_addr, local_addr_t
                          work_addr, local_addr_t coeff_addr, const dim4 *shape)
```

$$\text{dst}(n, c, h, w) = \text{arcsin}(\text{src}(n, c, h, w))$$

参数

- dst_addr -dst 的地址
- src_addr -src 的地址
- work_addr -work 的地址
- coeff_addr -coeff 的地址
- shape -dst、src 和 work 的 shape 的指针

注意事项

- dst、src 和 work 从同一个 NPU 开始，都是 64-byte aligned layout。
- coeff 通过 `tpu_bdc_load_fp32_arcsin_coeff()` 加载。
- work 存放中间结果，不允许 dst_addr 等于 work_addr，允许 src_addr 等于 dst_addr 或 work_addr，此情况下 src 的数据会被覆盖。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- src 的元素的取值范围是 [-1, 1]，dst 的元素的范围是 $[-\frac{\pi}{2}, \frac{\pi}{2}]$ 。

4.9.21 tpu_bdc_fp32_arccos

对张量的元素做 arccos 运算。

```
void tpu_bdc_fp32_arccos(local_addr_t dst_addr, local_addr_t src_addr, local_addr_t
                        work_addr, local_addr_t coeff_addr, const dim4 *shape)
```

$$\text{dst}(n, c, h, w) = \arccos(\text{src}(n, c, h, w))$$

参数

- dst_addr -dst 的地址
- src_addr -src 的地址
- work_addr -work 的地址
- coeff_addr -coeff 的地址
- shape -dst、src 和 work 的 shape 的指针

注意事项

- dst、src 和 work 从同一个 NPU 开始，都是 64-byte aligned layout。
- coeff 通过 `tpu_bdc_load_fp32_arcsin_coeff()` 加载。
- work 存放中间结果，不允许 dst_addr 等于 work_addr，允许 src_addr 等于 dst_addr 或 work_addr，此情况下 src 的数据会被覆盖。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- src 的元素的取值范围是 [-1, 1]，dst 的元素的范围是 $[0, \pi]$ 。

4.10 二元操作

4.10.1 tpu_bdc_and

两个张量的元素做按位与运算。

```
void tpu_bdc_and(local_addr_t dst_addr, local_addr_t src0_addr, local_addr_t
                src1_addr, const dim4 *shape, const dim4 *dst_stride, const dim4
                *src0_stride, const dim4 *src1_stride, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src0}(n, c, h, w) \text{ AND } \text{src1}(n, c, h, w)$$

参数

- dst_addr -dst 的地址
- src0_addr -src0 的地址
- src1_addr -src1 的地址
- shape -指向 dst、src0 和 src1 的 shape 的指针
- dst_stride -指向 dst 的 stride 的指针
- src0_stride -指向 src0 的 stride 的指针
- src1_stride -指向 src1 的 stride 的指针
- dtype -dst、src0 和 src1 的元素的数据类型

注意事项

- dst、src0 和 src1 从同一个 NPU 开始。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。

4.10.2 tpu_bdc_and_C

张量的元素与常数做按位与运算。

```
void tpu_bdc_and_C(local_addr_t dst_addr, local_addr_t src_addr, scalar_t C, const
                  dim4 *shape, const dim4 *dst_stride, const dim4 *src_stride,
                  data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(n, c, h, w) \text{ AND } C$$

参数

- dst_addr -dst 的地址
- src_addr -src 的地址
- C -常数
- shape -指向 dst 和 src 的 shape 的指针
- dst_stride -指向 dst 的 stride 的指针
- src_stride -指向 src 的 stride 的指针
- dtype -dst 和 src 的元素和 C 的数据类型

注意事项

- dst 和 src 从同一个 NPU 开始。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。

4.10.3 tpu_bdc_or

两个张量的元素做按位或运算。

```
void tpu_bdc_or(local_addr_t dst_addr, local_addr_t src0_addr, local_addr_t
                src1_addr, const dim4 *shape, const dim4 *dst_stride, const dim4
                *src0_stride, const dim4 *src1_stride, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src0}(n, c, h, w) \text{ OR } \text{src1}(n, c, h, w)$$

参数

- dst_addr -dst 的地址
- src0_addr -src0 的地址
- src1_addr -src1 的地址
- shape -指向 dst、src0 和 src1 的 shape 的指针
- dst_stride -指向 dst 的 stride 的指针
- src0_stride -指向 src0 的 stride 的指针

- `src1_stride` - 指向 `src1` 的 `stride` 的指针
- `dtype` - `dst`、`src0` 和 `src1` 的元素的数据类型

注意事项

- `dst`、`src0` 和 `src1` 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。
- 如果张量的 `stride` 指针是 `NULL`，则此张量是 64-byte aligned layout，否则是 free layout。

4.10.4 tpu_bdc_or_C

张量的元素与常数做按位或运算。

```
void tpu_bdc_or_C(local_addr_t dst_addr, local_addr_t src_addr, scalar_t C, const
                  dim4 *shape, const dim4 *dst_stride, const dim4 *src_stride,
                  data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(n, c, h, w) \text{ OR } C$$

参数

- `dst_addr` - `dst` 的地址
- `src_addr` - `src` 的地址
- `C` - 常数
- `shape` - 指向 `dst` 和 `src` 的 `shape` 的指针
- `dst_stride` - 指向 `dst` 的 `stride` 的指针
- `src_stride` - 指向 `src` 的 `stride` 的指针
- `dtype` - `dst` 和 `src` 的元素和 `C` 的数据类型

注意事项

- `dst` 和 `src` 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。
- 如果张量的 `stride` 指针是 `NULL`，则此张量是 64-byte aligned layout，否则是 free layout。

4.10.5 tpu_bdc_xor

两个张量的元素做按位异或运算。

```
void tpu_bdc_xor(local_addr_t dst_addr, local_addr_t src0_addr, local_addr_t
                  src1_addr, const dim4 *shape, const dim4 *dst_stride, const dim4
                  *src0_stride, const dim4 *src1_stride, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src0}(n, c, h, w) \text{ XOR } \text{src1}(n, c, h, w)$$

参数

- `dst_addr` - `dst` 的地址
- `src0_addr` - `src0` 的地址
- `src1_addr` - `src1` 的地址

- `shape` - 指向 `dst`、`src0` 和 `src1` 的 `shape` 的指针
- `dst_stride` - 指向 `dst` 的 `stride` 的指针
- `src0_stride` - 指向 `src0` 的 `stride` 的指针
- `src1_stride` - 指向 `src1` 的 `stride` 的指针
- `dtype` - `dst`、`src0` 和 `src1` 的元素的数据类型

注意事项

- `dst`、`src0` 和 `src1` 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 `[1, 65535]`。
- 如果张量的 `stride` 指针是 `NULL`，则此张量是 64-byte aligned layout，否则是 free layout。

4.10.6 tpu_bdc_xor_C

张量的元素与常数做按位异或运算。

```
void tpu_bdc_xor_C(local_addr_t dst_addr, local_addr_t src_addr, scalar_t C, const
                  dim4 *shape, const dim4 *dst_stride, const dim4 *src_stride,
                  data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(n, c, h, w) \text{ XOR } C$$

参数

- `dst_addr` - `dst` 的地址
- `src_addr` - `src` 的地址
- `C` - 常数
- `shape` - 指向 `dst` 和 `src` 的 `shape` 的指针
- `dst_stride` - 指向 `dst` 的 `stride` 的指针
- `src_stride` - 指向 `src` 的 `stride` 的指针
- `dtype` - `dst` 和 `src` 的元素和 `C` 的数据类型

注意事项

- `dst` 和 `src` 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 `[1, 65535]`。
- 如果张量的 `stride` 指针是 `NULL`，则此张量是 64-byte aligned layout，否则是 free layout。

4.10.7 tpu_bdc_min

两个张量的元素做取小运算。

```
void tpu_bdc_min(local_addr_t dst_addr, local_addr_t src0_addr, local_addr_t
                 src1_addr, const dim4 *shape, const dim4 *dst_stride, const dim4
                 *src0_stride, const dim4 *src1_stride, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \min(\text{src0}(n, c, h, w), \text{src1}(n, c, h, w))$$

参数

- `dst_addr` -dst 的地址
- `src0_addr` -src0 的地址
- `src1_addr` -src1 的地址
- `shape` -指向 dst、src0 和 src1 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针
- `src0_stride` -指向 src0 的 stride 的指针
- `src1_stride` -指向 src1 的 stride 的指针
- `dtype` -dst、src0 和 src1 的元素数据类型

注意事项

- dst、src0 和 src1 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。

4.10.8 tpu_bdc_min_C

张量的元素与常数做取小运算。

```
void tpu_bdc_min_C(local_addr_t dst_addr, local_addr_t src_addr, scalar_t C, const
                  dim4 *shape, const dim4 *dst_stride, const dim4 *src_stride,
                  data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \min(\text{src}(n, c, h, w), C)$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `C` -常数
- `shape` -指向 dst 和 src 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针
- `src_stride` -指向 src 的 stride 的指针
- `dtype` -dst 和 src 的元素和 C 的数据类型

注意事项

- dst 和 src 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。

4.10.9 tpu_bdc_max

两个张量的元素做取大运算。

```
void tpu_bdc_max(local_addr_t dst_addr, local_addr_t src0_addr, local_addr_t
                 src1_addr, const dim4 *shape, const dim4 *dst_stride, const dim4
                 *src0_stride, const dim4 *src1_stride, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \max(\text{src0}(n, c, h, w), \text{src1}(n, c, h, w))$$

参数

- dst_addr –dst 的地址
- src0_addr –src0 的地址
- src1_addr –src1 的地址
- shape –指向 dst、src0 和 src1 的 shape 的指针
- dst_stride –指向 dst 的 stride 的指针
- src0_stride –指向 src0 的 stride 的指针
- src1_stride –指向 src1 的 stride 的指针
- dtype –dst、src0 和 src1 的元素数据类型

注意事项

- dst、src0 和 src1 从同一个 NPU 开始。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。

4.10.10 tpu_bdc_max_C

张量的元素与常数做取大运算。

```
void tpu_bdc_max_C(local_addr_t dst_addr, local_addr_t src_addr, scalar_t C, const
                   dim4 *shape, const dim4 *dst_stride, const dim4 *src_stride,
                   data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \max(\text{src}(n, c, h, w), C)$$

参数

- dst_addr –dst 的地址
- src_addr –src 的地址
- C –常数
- shape –指向 dst 和 src 的 shape 的指针
- dst_stride –指向 dst 的 stride 的指针
- src_stride –指向 src 的 stride 的指针
- dtype –dst 和 src 的元素和 C 的数据类型

注意事项

- dst 和 src 从同一个 NPU 开始。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。

- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。

4.10.11 tpu_bdc_arithmetic_shift

张量的元素做算术移位运算。

```
void tpu_bdc_arithmetic_shift(local_addr_t dst_addr, local_addr_t src_addr,
                             local_addr_t shift_addr, const dim4 *shape, const dim4
                             *dst_stride, const dim4 *src_stride, const dim4
                             *shift_stride, data_type_t dst_dtype, data_type_t
                             src_dtype, data_type_t shift_dtype, rounding_mode_t
                             rounding_mode)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{src}(n, c, h, w) \text{ 左移 } \text{shift}(n, c, h, w) & \text{如果 } \text{shift}(n, c, h, w) > 0 \\ \text{src}(n, c, h, w) \text{ 右移 } -\text{shift}(n, c, h, w) & \text{其他情况} \end{cases}$$

参数

- dst_addr -dst 的地址
- src_addr -src 的地址
- shift_addr -shift 的地址
- shape -指向 dst、src 和 shift 的 shape 的指针
- dst_stride -指向 dst 的 stride 的指针
- src_stride -指向 src 的 stride 的指针
- shift_stride -指向 shift 的 stride 的指针
- dst_dtype -dst 的元素的数据类型
- src_dtype -src 的元素的数据类型
- shift_dtype -shift 的元素的数据类型
- rounding_mode -右移舍入模式

注意事项

- dst、src 和 shift 从同一个 NPU 开始。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- dst_dtype 和 src_dtype 的有效取值是 DT_INT32、DT_UINT32、DT_INT16、DT_UINT16、DT_INT8 和 DT_UINT8，同为有符号或同为无符号。
- shift_dtype 的有效取值是 DT_INT32、DT_INT16 和 DT_INT8，shift 的元素的取值范围是 [-32, 31]。
- src_dtype 的位宽大于等于 shift_dtype 的位宽。

4.10.12 tpu_bdc_arithmetic_shift_C

张量的元素做算术移位运算。

```
void tpu_bdc_arithmetic_shift_C(local_addr_t dst_addr, local_addr_t src_addr, char
                                C, const dim4 *shape, const dim4 *dst_stride, const
                                dim4 *src_stride, data_type_t dst_dtype,
                                data_type_t src_dtype, rounding_mode_t
                                rounding_mode)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{src}(n, c, h, w) \text{ 左移 } C & \text{如果 } C > 0 \\ \text{src}(n, c, h, w) \text{ 右移 } -C & \text{其他情况} \end{cases}$$

参数

- dst_addr -dst 的地址
- src_addr -src 的地址
- C -常数
- shape -指向 dst 和 src 的 shape 的指针
- dst_stride -指向 dst 的 stride 的指针
- src_stride -指向 src 的 stride 的指针
- dst_dtype -dst 的元素的数据类型
- src_dtype -src 的元素的数据类型
- rounding_mode -右移舍入模式

注意事项

- dst 和 src 从同一个 NPU 开始。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- dst_dtype 和 src_dtype 的有效取值是 DT_INT32、DT_UINT32、DT_INT16、DT_UINT16、DT_INT8 和 DT_UINT8，同为有符号或同为无符号。
- C 的取值范围是 [-32, 31]。

4.10.13 tpu_bdc_logical_shift

张量的元素做逻辑移位运算。

```
void tpu_bdc_logical_shift(local_addr_t dst_addr, local_addr_t src_addr,
                           local_addr_t shift_addr, const dim4 *shape, const dim4
                           *dst_stride, const dim4 *src_stride, const dim4 *shift_stride,
                           data_type_t dst_dtype, data_type_t src_dtype,
                           data_type_t shift_dtype, rounding_mode_t
                           rounding_mode)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{src}(n, c, h, w) \text{ 左移 } \text{shift}(n, c, h, w) & \text{如果 } \text{shift}(n, c, h, w) > 0 \\ \text{src}(n, c, h, w) \text{ 右移 } -\text{shift}(n, c, h, w) & \text{其他情况} \end{cases}$$

参数

- dst_addr -dst 的地址

- `src_addr` -src 的地址
- `shift_addr` -shift 的地址
- `shape` -指向 dst、src 和 shift 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针
- `src_stride` -指向 src 的 stride 的指针
- `shift_stride` -指向 shift 的 stride 的指针
- `dst_dtype` -dst 的元素的数据类型
- `src_dtype` -src 的元素的数据类型
- `shift_dtype` -shift 的元素的数据类型
- `rounding_mode` -右移舍入模式

注意事项

- dst、src 和 shift 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- `dst_dtype` 和 `src_dtype` 的有效取值是 `DT_UINT32`、`DT_UINT16` 和 `DT_UINT8`。
- `shift_dtype` 的有效取值是 `DT_INT32`、`DT_INT16` 和 `DT_INT8`，shift 的元素的取值范围是 [-32, 31]。
- `src_dtype` 的位宽大于等于 `shift_dtype` 的位宽。

4.10.14 tpu_bdc_logical_shift_C

张量的元素做逻辑移位运算。

```
void tpu_bdc_logical_shift_C(local_addr_t dst_addr, local_addr_t src_addr, char C,
                             const dim4 *shape, const dim4 *dst_stride, const dim4
                             *src_stride, data_type_t dst_dtype, data_type_t
                             src_dtype, rounding_mode_t rounding_mode)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{src}(n, c, h, w) \text{ 左移 } C & \text{如果 } C > 0 \\ \text{src}(n, c, h, w) \text{ 右移 } -C & \text{其他情况} \end{cases}$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `C` -常数
- `shape` -指向 dst 和 src 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针
- `src_stride` -指向 src 的 stride 的指针
- `dst_dtype` -dst 的元素的数据类型
- `src_dtype` -src 的元素的数据类型
- `rounding_mode` -右移舍入模式

注意事项

- dst 和 src 从同一个 NPU 开始。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- dst_dtype 和 src_dtype 的有效取值是 DT_UINT32、DT_UINT16 和 DT_UINT8。
- C 的取值范围是 [-32, 31]。

4.10.15 tpu_bdc_greater

比较张量的元素是否大于另一个张量的元素，可自定义真值。

```
void tpu_bdc_greater(local_addr_t dst_addr, local_addr_t src0_addr, local_addr_t
                    src1_addr, scalar_t true_val, const dim4 *shape, const dim4
                    *dst_stride, const dim4 *src0_stride, const dim4 *src1_stride,
                    data_type_t dst_dtype, data_type_t src_dtype)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{true_val} & \text{如果 } \text{src0}(n, c, h, w) > \text{src1}(n, c, h, w) \\ 0 & \text{其他情况} \end{cases}$$

参数

- dst_addr -dst 的地址
- src0_addr -src0 的地址
- src1_addr -src1 的地址
- true_val -真值
- shape -指向 dst、src0 和 src1 的 shape 的指针
- dst_stride -指向 dst 的 stride 的指针
- src0_stride -指向 src0 的 stride 的指针
- src1_stride -指向 src1 的 stride 的指针
- dst_dtype -dst 的元素和 true_val 的数据类型
- src_dtype -src 的元素的数据类型

注意事项

- dst、src0 和 src1 从同一个 NPU 开始。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- src_dtype 的位宽大于等于 dst_dtype 的位宽。

4.10.16 tpu_bdc_greater_C

比较张量的元素是否大于常数，可自定义真值。

```
void tpu_bdc_greater_C(local_addr_t dst_addr, local_addr_t src_addr, scalar_t C,
                      scalar_t true_val, const dim4 *shape, const dim4 *dst_stride,
                      const dim4 *src_stride, data_type_t dst_dtype, data_type_t
                      src_dtype)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{true_val} & \text{如果 } \text{src}(n, c, h, w) > C \\ 0 & \text{其他情况} \end{cases}$$

参数

- dst_addr -dst 的地址
- src_addr -src 的地址
- C -常数
- true_val -真值
- shape -指向 dst 和 src 的 shape 的指针
- dst_stride -指向 dst 的 stride 的指针
- src_stride -指向 src 的 stride 的指针
- dst_dtype -dst 的元素和 true_val 的数据类型
- src_dtype -src 的元素和 C 的数据类型

注意事项

- dst 和 src 从同一个 NPU 开始。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- src_dtype 的位宽大于等于 dst_dtype 的位宽。

4.10.17 tpu_bdc_less

比较张量的元素是否小于另一个张量的元素，可自定义真值。

```
void tpu_bdc_less(local_addr_t dst_addr, local_addr_t src0_addr, local_addr_t
                  src1_addr, scalar_t true_val, const dim4 *shape, const dim4
                  *dst_stride, const dim4 *src0_stride, const dim4 *src1_stride,
                  data_type_t dst_dtype, data_type_t src_dtype)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{true_val} & \text{如果 } \text{src0}(n, c, h, w) < \text{src1}(n, c, h, w) \\ 0 & \text{其他情况} \end{cases}$$

参数

- dst_addr -dst 的地址
- src0_addr -src0 的地址
- src1_addr -src1 的地址
- true_val -真值
- shape -指向 dst、src0 和 src1 的 shape 的指针

- `dst_stride` - 指向 `dst` 的 `stride` 的指针
- `src0_stride` - 指向 `src0` 的 `stride` 的指针
- `src1_stride` - 指向 `src1` 的 `stride` 的指针
- `dst_dtype` - `dst` 的元素和 `true_val` 的数据类型
- `src_dtype` - `src` 的元素的类型

注意事项

- `dst`、`src0` 和 `src1` 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 `[1, 65535]`。
- 如果张量的 `stride` 指针是 `NULL`，则此张量是 64-byte aligned layout，否则是 free layout。
- `src_dtype` 的位宽大于等于 `dst_dtype` 的位宽。

4.10.18 tpu_bdc_less_C

比较张量的元素是否小于常数，可自定义真值。

```
void tpu_bdc_less_C(local_addr_t dst_addr, local_addr_t src_addr, scalar_t C,
                    scalar_t true_val, const dim4 *shape, const dim4 *dst_stride,
                    const dim4 *src_stride, data_type_t dst_dtype, data_type_t
                    src_dtype)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{true_val} & \text{如果 } \text{src}(n, c, h, w) < C \\ 0 & \text{其他情况} \end{cases}$$

参数

- `dst_addr` - `dst` 的地址
- `src_addr` - `src` 的地址
- `C` - 常数
- `true_val` - 真值
- `shape` - 指向 `dst` 和 `src` 的 `shape` 的指针
- `dst_stride` - 指向 `dst` 的 `stride` 的指针
- `src_stride` - 指向 `src` 的 `stride` 的指针
- `dst_dtype` - `dst` 的元素和 `true_val` 的数据类型
- `src_dtype` - `src` 的元素和 `C` 的数据类型

注意事项

- `dst` 和 `src` 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 `[1, 65535]`。
- 如果张量的 `stride` 指针是 `NULL`，则此张量是 64-byte aligned layout，否则是 free layout。
- `src_dtype` 的位宽大于等于 `dst_dtype` 的位宽。

4.10.19 tpu_bdc_equal

比较张量的元素是否等于另一个张量的元素，可自定义真值。

```
void tpu_bdc_equal(local_addr_t dst_addr, local_addr_t src0_addr, local_addr_t
                  src1_addr, scalar_t true_val, const dim4 *shape, const dim4
                  *dst_stride, const dim4 *src0_stride, const dim4 *src1_stride,
                  data_type_t dst_dtype, data_type_t src_dtype)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{true_val} & \text{如果 } \text{src0}(n, c, h, w) = \text{src1}(n, c, h, w) \\ 0 & \text{其他情况} \end{cases}$$

参数

- dst_addr -dst 的地址
- src0_addr -src0 的地址
- src1_addr -src1 的地址
- true_val -真值
- shape -指向 dst、src0 和 src1 的 shape 的指针
- dst_stride -指向 dst 的 stride 的指针
- src0_stride -指向 src0 的 stride 的指针
- src1_stride -指向 src1 的 stride 的指针
- dst_dtype -dst 的元素和 true_val 的数据类型
- src_dtype -src 的元素的数据类型

注意事项

- dst、src0 和 src1 从同一个 NPU 开始。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- src_dtype 的位宽大于等于 dst_dtype 的位宽。

4.10.20 tpu_bdc_equal_C

比较张量的元素是否等于常数，可自定义真值。

```
void tpu_bdc_equal_C(local_addr_t dst_addr, local_addr_t src_addr, scalar_t C,
                    scalar_t true_val, const dim4 *shape, const dim4 *dst_stride,
                    const dim4 *src_stride, data_type_t dst_dtype, data_type_t
                    src_dtype)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{true_val} & \text{如果 } \text{src}(n, c, h, w) = C \\ 0 & \text{其他情况} \end{cases}$$

参数

- dst_addr -dst 的地址
- src_addr -src 的地址
- C -常数
- true_val -真值

- `shape` –指向 `dst` 和 `src` 的 `shape` 的指针
- `dst_stride` –指向 `dst` 的 `stride` 的指针
- `src_stride` –指向 `src` 的 `stride` 的指针
- `dst_dtype` –`dst` 的元素和 `true_val` 的数据类型
- `src_dtype` –`src` 的元素和 `C` 的数据类型

注意事项

- `dst` 和 `src` 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 `[1, 65535]`。
- 如果张量的 `stride` 指针是 `NULL`，则此张量是 64-byte aligned layout，否则是 free layout。
- `src_dtype` 的位宽大于等于 `dst_dtype` 的位宽。

4.10.21 tpu_bdc_greater_equal

比较张量的元素是否大于等于另一个张量的元素，可自定义真值。

```
void tpu_bdc_greater_equal(local_addr_t dst_addr, local_addr_t src0_addr,
                          local_addr_t src1_addr, scalar_t true_val, const dim4
                          *shape, const dim4 *dst_stride, const dim4 *src0_stride,
                          const dim4 *src1_stride, data_type_t dst_dtype,
                          data_type_t src_dtype)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{true_val} & \text{如果 } \text{src0}(n, c, h, w) \geq \text{src1}(n, c, h, w) \\ 0 & \text{其他情况} \end{cases}$$

参数

- `dst_addr` –`dst` 的地址
- `src0_addr` –`src0` 的地址
- `src1_addr` –`src1` 的地址
- `true_val` –真值
- `shape` –指向 `dst`、`src0` 和 `src1` 的 `shape` 的指针
- `dst_stride` –指向 `dst` 的 `stride` 的指针
- `src0_stride` –指向 `src0` 的 `stride` 的指针
- `src1_stride` –指向 `src1` 的 `stride` 的指针
- `dst_dtype` –`dst` 的元素和 `true_val` 的数据类型
- `src_dtype` –`src` 的元素的的数据类型

注意事项

- `dst`、`src0` 和 `src1` 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 `[1, 65535]`。
- 如果张量的 `stride` 指针是 `NULL`，则此张量是 64-byte aligned layout，否则是 free layout。
- `src_dtype` 的位宽大于等于 `dst_dtype` 的位宽。

4.10.22 tpu_bdc_greater_equal_C

比较张量的元素是否大于等于常数，可自定义真值。

```
void tpu_bdc_greater_equal_C(local_addr_t dst_addr, local_addr_t src_addr,
                             scalar_t C, scalar_t true_val, const dim4 *shape, const
                             dim4 *dst_stride, const dim4 *src_stride, data_type_t
                             dst_dtype, data_type_t src_dtype)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{true_val} & \text{如果 } \text{src}(n, c, h, w) \geq C \\ 0 & \text{其他情况} \end{cases}$$

参数

- dst_addr -dst 的地址
- src_addr -src 的地址
- C -常数
- true_val -真值
- shape -指向 dst 和 src 的 shape 的指针
- dst_stride -指向 dst 的 stride 的指针
- src_stride -指向 src 的 stride 的指针
- dst_dtype -dst 的元素和 true_val 的数据类型
- src_dtype -src 的元素和 C 的数据类型

注意事项

- dst 和 src 从同一个 NPU 开始。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- src_dtype 的位宽大于等于 dst_dtype 的位宽。

4.10.23 tpu_bdc_less_equal

比较张量的元素是否小于等于另一个张量的元素，可自定义真值。

```
void tpu_bdc_less_equal(local_addr_t dst_addr, local_addr_t src0_addr, local_addr_t
                        src1_addr, scalar_t true_val, const dim4 *shape, const dim4
                        *dst_stride, const dim4 *src0_stride, const dim4 *src1_stride,
                        data_type_t dst_dtype, data_type_t src0_dtype, data_type_t src1_dtype)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{true_val} & \text{如果 } \text{src0}(n, c, h, w) \leq \text{src1}(n, c, h, w) \\ 0 & \text{其他情况} \end{cases}$$

参数

- dst_addr -dst 的地址
- src0_addr -src0 的地址
- src1_addr -src1 的地址
- true_val -真值
- shape -指向 dst、src0 和 src1 的 shape 的指针

- `dst_stride` - 指向 `dst` 的 `stride` 的指针
- `src0_stride` - 指向 `src0` 的 `stride` 的指针
- `src1_stride` - 指向 `src1` 的 `stride` 的指针
- `dst_dtype` - `dst` 的元素和 `true_val` 的数据类型
- `src_dtype` - `src` 的元素的类型

注意事项

- `dst`、`src0` 和 `src1` 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 `[1, 65535]`。
- 如果张量的 `stride` 指针是 `NULL`，则此张量是 64-byte aligned layout，否则是 free layout。
- `src_dtype` 的位宽大于等于 `dst_dtype` 的位宽。

4.10.24 tpu_bdc_less_equal_C

比较张量的元素是否小于等于常数，可自定义真值。

```
void tpu_bdc_less_equal_C(local_addr_t dst_addr, local_addr_t src_addr, scalar_t C,
                          scalar_t true_val, const dim4 *shape, const dim4
                          *dst_stride, const dim4 *src_stride, data_type_t
                          dst_dtype, data_type_t src_dtype)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{true_val} & \text{如果 } \text{src}(n, c, h, w) \leq C \\ 0 & \text{其他情况} \end{cases}$$

参数

- `dst_addr` - `dst` 的地址
- `src_addr` - `src` 的地址
- `C` - 常数
- `true_val` - 真值
- `shape` - 指向 `dst` 和 `src` 的 `shape` 的指针
- `dst_stride` - 指向 `dst` 的 `stride` 的指针
- `src_stride` - 指向 `src` 的 `stride` 的指针
- `dst_dtype` - `dst` 的元素和 `true_val` 的数据类型
- `src_dtype` - `src` 的元素和 `C` 的数据类型

注意事项

- `dst` 和 `src` 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 `[1, 65535]`。
- 如果张量的 `stride` 指针是 `NULL`，则此张量是 64-byte aligned layout，否则是 free layout。
- `src_dtype` 的位宽大于等于 `dst_dtype` 的位宽。

4.10.25 tpu_bdc_not_equal

比较张量的元素是否不等于另一个张量的元素，可自定义真值。

```
void tpu_bdc_not_equal(local_addr_t dst_addr, local_addr_t src0_addr, local_addr_t
                      src1_addr, scalar_t true_val, const dim4 *shape, const dim4
                      *dst_stride, const dim4 *src0_stride, const dim4 *src1_stride,
                      data_type_t dst_dtype, data_type_t src_dtype)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{true_val} & \text{如果 } \text{src0}(n, c, h, w) \neq \text{src1}(n, c, h, w) \\ 0 & \text{其他情况} \end{cases}$$

参数

- dst_addr –dst 的地址
- src0_addr –src0 的地址
- src1_addr –src1 的地址
- true_val –真值
- shape –指向 dst、src0 和 src1 的 shape 的指针
- dst_stride –指向 dst 的 stride 的指针
- src0_stride –指向 src0 的 stride 的指针
- src1_stride –指向 src1 的 stride 的指针
- dst_dtype –dst 的元素和 true_val 的数据类型
- src_dtype –src 的元素的数据类型

注意事项

- dst、src0 和 src1 从同一个 NPU 开始。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- src_dtype 的位宽大于等于 dst_dtype 的位宽。

4.10.26 tpu_bdc_not_equal_C

比较张量的元素是否不等于常数，可自定义真值。

```
void tpu_bdc_not_equal_C(local_addr_t dst_addr, local_addr_t src_addr, scalar_t C,
                         scalar_t true_val, const dim4 *shape, const dim4
                         *dst_stride, const dim4 *src_stride, data_type_t
                         dst_dtype, data_type_t src_dtype)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{true_val} & \text{如果 } \text{src}(n, c, h, w) \neq C \\ 0 & \text{其他情况} \end{cases}$$

参数

- dst_addr –dst 的地址
- src_addr –src 的地址
- C –常数
- true_val –真值

- `shape` - 指向 `dst` 和 `src` 的 `shape` 的指针
- `dst_stride` - 指向 `dst` 的 `stride` 的指针
- `src_stride` - 指向 `src` 的 `stride` 的指针
- `dst_dtype` - `dst` 的元素和 `true_val` 的数据类型
- `src_dtype` - `src` 的元素和 `C` 的数据类型

注意事项

- `dst` 和 `src` 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 `[1, 65535]`。
- 如果张量的 `stride` 指针是 `NULL`，则此张量是 64-byte aligned layout，否则是 free layout。
- `src_dtype` 的位宽大于等于 `dst_dtype` 的位宽。

4.10.27 tpu_bdc_vc_and

两个向量的元素交叉做按位与运算。

```
void tpu_bdc_vc_and(local_addr_t dst_addr, local_addr_t src0_addr, local_addr_t
                    src1_addr, int src0_len, int src1_len, int src0_len_per_channel,
                    int src1_len_per_channel, data_type_t dtype)
```

$$\text{dst}(m, n) = \text{src0}(m) \text{ AND } \text{src1}(n)$$

参数

- `dst_addr` - `dst` 的地址
- `src0_addr` - `src0` 的地址
- `src1_addr` - `src1` 的地址
- `src0_len` - `src0` 的长度
- `src1_len` - `src1` 的长度
- `src0_len_per_channel` - `src0` 在每个 channel 的长度
- `src1_len_per_channel` - `src1` 在每个 channel 的长度
- `dtype` - `dst`、`src0` 和 `src1` 的元素的 datatype

注意事项

- `dst` 和 `src1` 从同一个 NPU 开始。
- `src0_len` 和 `src1_len` 的取值范围是 `[1, 65535]`。
- `dst` 是 matrix layout，`dst` 的行数是 `src0_len`，列数是 `src1_len`，`src0` 和 `src1` 是 vector layout。

4.10.28 tpu_bdc_vc_or

两个向量的元素交叉做按位或运算。

```
void tpu_bdc_vc_or(local_addr_t dst_addr, local_addr_t src0_addr, local_addr_t
                  src1_addr, int src0_len, int src1_len, int src0_len_per_channel, int
                  src1_len_per_channel, data_type_t dtype)
```

$$\text{dst}(m, n) = \text{src0}(m) \text{ OR } \text{src1}(n)$$

参数

- dst_addr –dst 的地址
- src0_addr –src0 的地址
- src1_addr –src1 的地址
- src0_len –src0 的长度
- src1_len –src1 的长度
- src0_len_per_channel –src0 在每个 channel 的长度
- src1_len_per_channel –src1 在每个 channel 的长度
- dtype –dst、src0 和 src1 的元素的数据类型

注意事项

- dst 和 src1 从同一个 NPU 开始。
- src0_len 和 src1_len 的取值范围是 [1, 65535]。
- dst 是 matrix layout, dst 的行数是 src0_len, 列数是 src1_len, src0 和 src1 是 vector layout。

4.10.29 tpu_bdc_vc_xor

两个向量的元素交叉做按位异或运算。

```
void tpu_bdc_vc_xor(local_addr_t dst_addr, local_addr_t src0_addr, local_addr_t
                  src1_addr, int src0_len, int src1_len, int src0_len_per_channel,
                  int src1_len_per_channel, data_type_t dtype)
```

$$\text{dst}(m, n) = \text{src0}(m) \text{ XOR } \text{src1}(n)$$

参数

- dst_addr –dst 的地址
- src0_addr –src0 的地址
- src1_addr –src1 的地址
- src0_len –src0 的长度
- src1_len –src1 的长度
- src0_len_per_channel –src0 在每个 channel 的长度
- src1_len_per_channel –src1 在每个 channel 的长度
- dtype –dst、src0 和 src1 的元素的数据类型

注意事项

- dst 和 src1 从同一个 NPU 开始。

- `src0_len` 和 `src1_len` 的取值范围是 [1, 65535]。
- `dst` 是 matrix layout, `dst` 的行数是 `src0_len`, 列数是 `src1_len`, `src0` 和 `src1` 是 vector layout。

4.10.30 tpu_bdc_vc_min

两个向量的元素交叉做取小运算。

```
void tpu_bdc_vc_min(local_addr_t dst_addr, local_addr_t src0_addr, local_addr_t
                    src1_addr, int src0_len, int src1_len, int src0_len_per_channel,
                    int src1_len_per_channel, data_type_t dtype)
```

$$\text{dst}(m, n) = \min(\text{src0}(m), \text{src1}(n))$$

参数

- `dst_addr` - `dst` 的地址
- `src0_addr` - `src0` 的地址
- `src1_addr` - `src1` 的地址
- `src0_len` - `src0` 的长度
- `src1_len` - `src1` 的长度
- `src0_len_per_channel` - `src0` 在每个 channel 的长度
- `src1_len_per_channel` - `src1` 在每个 channel 的长度
- `dtype` - `dst`、`src0` 和 `src1` 的元素的数据类型

注意事项

- `dst` 和 `src1` 从同一个 NPU 开始。
- `src0_len` 和 `src1_len` 的取值范围是 [1, 65535]。
- `dst` 是 matrix layout, `dst` 的行数是 `src0_len`, 列数是 `src1_len`, `src0` 和 `src1` 是 vector layout。

4.10.31 tpu_bdc_vc_max

两个向量的元素交叉做取大运算。

```
void tpu_bdc_vc_max(local_addr_t dst_addr, local_addr_t src0_addr, local_addr_t
                    src1_addr, int src0_len, int src1_len, int src0_len_per_channel,
                    int src1_len_per_channel, data_type_t dtype)
```

$$\text{dst}(m, n) = \max(\text{src0}(m), \text{src1}(n))$$

参数

- `dst_addr` - `dst` 的地址
- `src0_addr` - `src0` 的地址
- `src1_addr` - `src1` 的地址
- `src0_len` - `src0` 的长度
- `src1_len` - `src1` 的长度
- `src0_len_per_channel` - `src0` 在每个 channel 的长度

- `src1_len_per_channel` -src1 在每个 channel 的长度
- `dtype` -dst、src0 和 src1 的元素的数据类型

注意事项

- dst 和 src1 从同一个 NPU 开始。
- `src0_len` 和 `src1_len` 的取值范围是 [1, 65535]。
- dst 是 matrix layout, dst 的行数是 `src0_len`, 列数是 `src1_len`, src0 和 src1 是 vector layout。

4.10.32 tpu_bdc_vc_greater

交叉比较向量的元素是否大于另一个张量的元素，可自定义真值。

```
void tpu_bdc_vc_greater(local_addr_t dst_addr, local_addr_t src0_addr, local_addr_t
                        src1_addr, scalar_t true_val, int src0_len, int src1_len, int
                        src0_len_per_channel, int src1_len_per_channel,
                        data_type_t dst_dtype, data_type_t src_dtype)
```

$$\text{dst}(m, n) = \begin{cases} \text{true_val} & \text{如果 } \text{src0}(m) > \text{src1}(n) \\ 0 & \text{其他情况} \end{cases}$$

参数

- `dst_addr` -dst 的地址
- `src0_addr` -src0 的地址
- `src1_addr` -src1 的地址
- `true_val` -真值
- `src0_len` -src0 的长度
- `src1_len` -src1 的长度
- `src0_len_per_channel` -src0 在每个 channel 的长度
- `src1_len_per_channel` -src1 在每个 channel 的长度
- `dst_dtype` -dst 的元素和 `true_val` 的数据类型
- `src_dtype` -src 的元素的数据类型

注意事项

- dst 和 src1 从同一个 NPU 开始。
- `src0_len` 和 `src1_len` 的取值范围是 [1, 65535]。
- dst 是 matrix layout, dst 的行数是 `src0_len`, 列数是 `src1_len`, src0 和 src1 是 vector layout。
- `src_dtype` 的位宽大于等于 `dst_dtype` 的位宽。

4.10.33 tpu_bdc_vc_less

交叉比较向量的元素是否小于另一个张量的元素，可自定义真值。

```
void tpu_bdc_vc_less(local_addr_t dst_addr, local_addr_t src0_addr, local_addr_t
                    src1_addr, scalar_t true_val, int src0_len, int src1_len, int
                    src0_len_per_channel, int src1_len_per_channel, data_type_t
                    dst_dtype, data_type_t src_dtype)
```

$$dst(m, n) = \begin{cases} true_val & \text{如果 } src0(m) < src1(n) \\ 0 & \text{其他情况} \end{cases}$$

参数

- dst_addr -dst 的地址
- src0_addr -src0 的地址
- src1_addr -src1 的地址
- true_val -真值
- src0_len -src0 的长度
- src1_len -src1 的长度
- src0_len_per_channel -src0 在每个 channel 的长度
- src1_len_per_channel -src1 在每个 channel 的长度
- dst_dtype -dst 的元素和 true_val 的数据类型
- src_dtype -src 的元素的数据类型

注意事项

- dst 和 src1 从同一个 NPU 开始。
- src0_len 和 src1_len 的取值范围是 [1, 65535]。
- dst 是 matrix layout, dst 的行数是 src0_len, 列数是 src1_len, src0 和 src1 是 vector layout。
- src_dtype 的位宽大于等于 dst_dtype 的位宽。

4.10.34 tpu_bdc_vc_equal

交叉比较向量的元素是否等于另一个张量的元素，可自定义真值。

```
void tpu_bdc_vc_equal(local_addr_t dst_addr, local_addr_t src0_addr, local_addr_t
                    src1_addr, scalar_t true_val, int src0_len, int src1_len, int
                    src0_len_per_channel, int src1_len_per_channel, data_type_t
                    dst_dtype, data_type_t src_dtype)
```

$$dst(m, n) = \begin{cases} true_val & \text{如果 } src0(m) = src1(n) \\ 0 & \text{其他情况} \end{cases}$$

参数

- dst_addr -dst 的地址
- src0_addr -src0 的地址
- src1_addr -src1 的地址
- true_val -真值

- `src0_len` - `src0` 的长度
- `src1_len` - `src1` 的长度
- `src0_len_per_channel` - `src0` 在每个 channel 的长度
- `src1_len_per_channel` - `src1` 在每个 channel 的长度
- `dst_dtype` - `dst` 的元素和 `true_val` 的数据类型
- `src_dtype` - `src` 的元素的类型

注意事项

- `dst` 和 `src1` 从同一个 NPU 开始。
- `src0_len` 和 `src1_len` 的取值范围是 [1, 65535]。
- `dst` 是 matrix layout, `dst` 的行数是 `src0_len`, 列数是 `src1_len`, `src0` 和 `src1` 是 vector layout。
- `src_dtype` 的位宽大于等于 `dst_dtype` 的位宽。

4.10.35 tpu_bdc_vc_greater_equal

交叉比较向量的元素是否大于等于另一个张量的元素，可自定义真值。

```
void tpu_bdc_vc_greater_equal(local_addr_t dst_addr, local_addr_t src0_addr,
                             local_addr_t src1_addr, scalar_t true_val, int
                             src0_len, int src1_len, int src0_len_per_channel, int
                             src1_len_per_channel, data_type_t dst_dtype,
                             data_type_t src_dtype)
```

$$\text{dst}(m, n) = \begin{cases} \text{true_val} & \text{如果 } \text{src0}(m) \geq \text{src1}(n) \\ 0 & \text{其他情况} \end{cases}$$

参数

- `dst_addr` - `dst` 的地址
- `src0_addr` - `src0` 的地址
- `src1_addr` - `src1` 的地址
- `true_val` - 真值
- `src0_len` - `src0` 的长度
- `src1_len` - `src1` 的长度
- `src0_len_per_channel` - `src0` 在每个 channel 的长度
- `src1_len_per_channel` - `src1` 在每个 channel 的长度
- `dst_dtype` - `dst` 的元素和 `true_val` 的数据类型
- `src_dtype` - `src` 的元素的类型

注意事项

- `dst` 和 `src1` 从同一个 NPU 开始。
- `src0_len` 和 `src1_len` 的取值范围是 [1, 65535]。
- `dst` 是 matrix layout, `dst` 的行数是 `src0_len`, 列数是 `src1_len`, `src0` 和 `src1` 是 vector layout。
- `src_dtype` 的位宽大于等于 `dst_dtype` 的位宽。

4.10.36 tpu_bdc_vc_less_equal

交叉比较向量的元素是否小于等于另一个张量的元素，可自定义真值。

```
void tpu_bdc_vc_less_equal(local_addr_t dst_addr, local_addr_t src0_addr,
                           local_addr_t src1_addr, scalar_t true_val, int src0_len,
                           int src1_len, int src0_len_per_channel, int
                           src1_len_per_channel, data_type_t dst_dtype,
                           data_type_t src_dtype)
```

$$\text{dst}(m, n) = \begin{cases} \text{true_val} & \text{如果 } \text{src0}(m) \leq \text{src1}(n) \\ 0 & \text{其他情况} \end{cases}$$

参数

- dst_addr -dst 的地址
- src0_addr -src0 的地址
- src1_addr -src1 的地址
- true_val -真值
- src0_len -src0 的长度
- src1_len -src1 的长度
- src0_len_per_channel -src0 在每个 channel 的长度
- src1_len_per_channel -src1 在每个 channel 的长度
- dst_dtype -dst 的元素和 true_val 的数据类型
- src_dtype -src 的元素的数据类型

注意事项

- dst 和 src1 从同一个 NPU 开始。
- src0_len 和 src1_len 的取值范围是 [1, 65535]。
- dst 是 matrix layout, dst 的行数是 src0_len, 列数是 src1_len, src0 和 src1 是 vector layout。
- src_dtype 的位宽大于等于 dst_dtype 的位宽。

4.10.37 tpu_bdc_vc_not_equal

交叉比较向量的元素是否不等于另一个张量的元素，可自定义真值。

```
void tpu_bdc_vc_not_equal(local_addr_t dst_addr, local_addr_t src0_addr,
                           local_addr_t src1_addr, scalar_t true_val, int src0_len,
                           int src1_len, int src0_len_per_channel, int
                           src1_len_per_channel, data_type_t dst_dtype,
                           data_type_t src_dtype)
```

$$\text{dst}(m, n) = \begin{cases} \text{true_val} & \text{如果 } \text{src0}(m) \neq \text{src1}(n) \\ 0 & \text{其他情况} \end{cases}$$

参数

- dst_addr -dst 的地址
- src0_addr -src0 的地址

- `src1_addr` - `src1` 的地址
- `true_val` - 真值
- `src0_len` - `src0` 的长度
- `src1_len` - `src1` 的长度
- `src0_len_per_channel` - `src0` 在每个 channel 的长度
- `src1_len_per_channel` - `src1` 在每个 channel 的长度
- `dst_dtype` - `dst` 的元素和 `true_val` 的数据类型
- `src_dtype` - `src` 的元素的类型

注意事项

- `dst` 和 `src1` 从同一个 NPU 开始。
- `src0_len` 和 `src1_len` 的取值范围是 `[1, 65535]`。
- `dst` 是 matrix layout, `dst` 的行数是 `src0_len`, 列数是 `src1_len`, `src0` 和 `src1` 是 vector layout。
- `src_dtype` 的位宽大于等于 `dst_dtype` 的位宽。

4.11 浮点二元操作

4.11.1 tpu_bdc_fp_add

两个张量的元素相加。

```
void tpu_bdc_fp_add(local_addr_t dst_addr, local_addr_t src0_addr, local_addr_t
                    src1_addr, const dim4 *shape, const dim4 *dst_stride, const dim4
                    *src0_stride, const dim4 *src1_stride, data_type_t dtype)
```

$$dst(n, c, h, w) = src0(n, c, h, w) + src1(n, c, h, w)$$

参数

- `dst_addr` - `dst` 的地址
- `src0_addr` - `src0` 的地址
- `src1_addr` - `src1` 的地址
- `shape` - 指向 `dst`、`src0` 和 `src1` 的 `shape` 的指针
- `dst_stride` - 指向 `dst` 的 `stride` 的指针
- `src0_stride` - 指向 `src0` 的 `stride` 的指针
- `src1_stride` - 指向 `src1` 的 `stride` 的指针
- `dtype` - `dst`、`src0` 和 `src1` 的元素的类型

注意事项

- `dst`、`src0` 和 `src1` 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 `[1, 65535]`。
- 如果张量的 `stride` 指针是 `NULL`, 则此张量是 64-byte aligned layout, 否则是 free layout。
- `dtype` 的有效取值是 `DT_FP32`、`DT_FP16` 和 `DT_BFP16`。

4.11.2 tpu_bdc_fp_add_C

张量的元素和常数相加。

```
void tpu_bdc_fp_add_C(local_addr_t dst_addr, local_addr_t src_addr, scalar_t C,
                      const dim4 *shape, const dim4 *dst_stride, const dim4
                      *src_stride, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(n, c, h, w) + C$$

参数

- `dst_addr` –dst 的地址
- `src_addr` –src 的地址
- `C` –常数
- `shape` –指向 dst 和 src 的 shape 的指针
- `dst_stride` –指向 dst 的 stride 的指针
- `src_stride` –指向 src 的 stride 的指针
- `dtype` –dst 和 src 的元素和 `C` 的数据类型

注意事项

- dst 和 src 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- `dtype` 的有效取值是 `DT_FP32`、`DT_FP16` 和 `DT_BFP16`。

4.11.3 tpu_bdc_fp_sub

两个张量的元素相减。

```
void tpu_bdc_fp_sub(local_addr_t dst_addr, local_addr_t src0_addr, local_addr_t
                    src1_addr, const dim4 *shape, const dim4 *dst_stride, const dim4
                    *src0_stride, const dim4 *src1_stride, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src0}(n, c, h, w) - \text{src1}(n, c, h, w)$$

参数

- `dst_addr` –dst 的地址
- `src0_addr` –src0 的地址
- `src1_addr` –src1 的地址
- `shape` –指向 dst、src0 和 src1 的 shape 的指针
- `dst_stride` –指向 dst 的 stride 的指针
- `src0_stride` –指向 src0 的 stride 的指针
- `src1_stride` –指向 src1 的 stride 的指针
- `dtype` –dst、src0 和 src1 的元素的数据类型

注意事项

- dst、src0 和 src1 从同一个 NPU 开始。

- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 $[1, 65535]$ 。
- 如果张量的 `stride` 指针是 `NULL`，则此张量是 64-byte aligned layout，否则是 free layout。
- `dtype` 的有效取值是 `DT_FP32`、`DT_FP16` 和 `DT_BFP16`。

4.11.4 tpu_bdc_fp_sub_C

张量的元素减常数。

```
void tpu_bdc_fp_sub_C(local_addr_t dst_addr, local_addr_t src_addr, scalar_t C,
                      const dim4 *shape, const dim4 *dst_stride, const dim4
                      *src_stride, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(n, c, h, w) - C$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `C` -常数
- `shape` -指向 `dst` 和 `src` 的 `shape` 的指针
- `dst_stride` -指向 `dst` 的 `stride` 的指针
- `src_stride` -指向 `src` 的 `stride` 的指针
- `dtype` -dst 和 `src` 的元素和 `C` 的数据类型

注意事项

- `dst` 和 `src` 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 $[1, 65535]$ 。
- 如果张量的 `stride` 指针是 `NULL`，则此张量是 64-byte aligned layout，否则是 free layout。
- `dtype` 的有效取值是 `DT_FP32`、`DT_FP16` 和 `DT_BFP16`。

4.11.5 tpu_bdc_fp_C_sub

常数减张量的元素。

```
void tpu_bdc_fp_C_sub(local_addr_t dst_addr, local_addr_t src_addr, scalar_t C,
                      const dim4 *shape, const dim4 *dst_stride, const dim4
                      *src_stride, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = C - \text{src}(n, c, h, w)$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `C` -常数
- `shape` -指向 `dst` 和 `src` 的 `shape` 的指针
- `dst_stride` -指向 `dst` 的 `stride` 的指针

- `src_stride` - 指向 `src` 的 `stride` 的指针
- `dtype` - `dst` 和 `src` 的元素和 `C` 的数据类型

注意事项

- `dst` 和 `src` 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 `[1, 65535]`。
- 如果张量的 `stride` 指针是 `NULL`，则此张量是 64-byte aligned layout，否则是 free layout。
- `dtype` 的有效取值是 `DT_FP32`、`DT_FP16` 和 `DT_BFP16`。

4.11.6 tpu_bdc_fp_mul

两个张量的元素相乘。

```
void tpu_bdc_fp_mul(local_addr_t dst_addr, local_addr_t src0_addr, local_addr_t
                    src1_addr, const dim4 *shape, const dim4 *dst_stride, const dim4
                    *src0_stride, const dim4 *src1_stride, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src0}(n, c, h, w) \times \text{src1}(n, c, h, w)$$

参数

- `dst_addr` - `dst` 的地址
- `src0_addr` - `src0` 的地址
- `src1_addr` - `src1` 的地址
- `shape` - 指向 `dst`、`src0` 和 `src1` 的 `shape` 的指针
- `dst_stride` - 指向 `dst` 的 `stride` 的指针
- `src0_stride` - 指向 `src0` 的 `stride` 的指针
- `src1_stride` - 指向 `src1` 的 `stride` 的指针
- `dtype` - `dst`、`src0` 和 `src1` 的元素的类型

注意事项

- `dst`、`src0` 和 `src1` 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 `[1, 65535]`。
- 如果张量的 `stride` 指针是 `NULL`，则此张量是 64-byte aligned layout，否则是 free layout。
- `dtype` 的有效取值是 `DT_FP32`、`DT_FP16` 和 `DT_BFP16`。

4.11.7 tpu_bdc_fp_mul_C

张量的元素和常数相乘。

```
void tpu_bdc_fp_mul_C(local_addr_t dst_addr, local_addr_t src_addr, scalar_t C,
                      const dim4 *shape, const dim4 *dst_stride, const dim4
                      *src_stride, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(n, c, h, w) \times C$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `C` -常数
- `shape` -指向 `dst` 和 `src` 的 `shape` 的指针
- `dst_stride` -指向 `dst` 的 `stride` 的指针
- `src_stride` -指向 `src` 的 `stride` 的指针
- `dtype` -dst 和 `src` 的元素和 `C` 的数据类型

注意事项

- `dst` 和 `src` 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。
- 如果张量的 `stride` 指针是 `NULL`，则此张量是 64-byte aligned layout，否则是 free layout。
- `dtype` 的有效取值是 `DT_FP32`、`DT_FP16` 和 `DT_BFP16`。

4.11.8 tpu_bdc_fp32_div

两个张量的元素相除。

```
void tpu_bdc_fp32_div(local_addr_t dst_addr, local_addr_t src0_addr, local_addr_t
                      src1_addr, const dim4 *shape, const dim4 *dst_stride, const
                      dim4 *src0_stride, const dim4 *src1_stride)
```

$$\text{dst}(n, c, h, w) = \frac{\text{src0}(n, c, h, w)}{\text{src1}(n, c, h, w)}$$

参数

- `dst_addr` -dst 的地址
- `src0_addr` -src0 的地址
- `src1_addr` -src1 的地址
- `shape` -指向 `dst`、`src0` 和 `src1` 的 `shape` 的指针
- `dst_stride` -指向 `dst` 的 `stride` 的指针
- `src0_stride` -指向 `src0` 的 `stride` 的指针
- `src1_stride` -指向 `src1` 的 `stride` 的指针

注意事项

- `dst`、`src0` 和 `src1` 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。
- 如果张量的 `stride` 指针是 `NULL`，则此张量是 64-byte aligned layout，否则是 free layout。
- 等价于 Newton 迭代次数是 3 的 `tpu_bdc_fp32_tunable_div()`。

4.11.9 tpu_bdc_fp32_div_C

张量的元素除以常数。

```
void tpu_bdc_fp32_div_C(local_addr_t dst_addr, local_addr_t src_addr, float C,
                        const dim4 *shape, const dim4 *dst_stride, const dim4
                        *src_stride)
```

$$\text{dst}(n, c, h, w) = \frac{\text{src}(n, c, h, w)}{C}$$

参数

- dst_addr –dst 的地址
- src_addr –src 的地址
- C –常数
- shape –指向 dst 和 src 的 shape 的指针
- dst_stride –指向 dst 的 stride 的指针
- src_stride –指向 src 的 stride 的指针

注意事项

- dst 和 src 从同一个 NPU 开始。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- 等价于 Newton 迭代次数是 3 的 tpu_bdc_fp32_tunable_div_C()。

4.11.10 tpu_bdc_fp32_C_div

常数除以张量的元素。

```
void tpu_bdc_fp32_C_div(local_addr_t dst_addr, local_addr_t src_addr, float C,
                        const dim4 *shape, const dim4 *dst_stride, const dim4
                        *src_stride)
```

$$\text{dst}(n, c, h, w) = \frac{C}{\text{src}(n, c, h, w)}$$

参数

- dst_addr –dst 的地址
- src_addr –src 的地址
- C –常数
- shape –指向 dst 和 src 的 shape 的指针
- dst_stride –指向 dst 的 stride 的指针
- src_stride –指向 src 的 stride 的指针

注意事项

- dst 和 src 从同一个 NPU 开始。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。

- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- 等价于 Newton 迭代次数是 3 的 `tpu_bdc_fp32_tunable_C_div()`。

4.11.11 tpu_bdc_fp32_tunable_div

两个张量的元素相除。

```
void tpu_bdc_fp32_tunable_div(local_addr_t dst_addr, local_addr_t src0_addr,
                             local_addr_t src1_addr, const dim4 *shape, const dim4
                             *dst_stride, const dim4 *src0_stride, const dim4
                             *src1_stride, int num_iter)
```

$$\text{dst}(n, c, h, w) = \frac{\text{src0}(n, c, h, w)}{\text{src1}(n, c, h, w)}$$

参数

- `dst_addr` -dst 的地址
- `src0_addr` -src0 的地址
- `src1_addr` -src1 的地址
- `shape` -指向 dst、src0 和 src1 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针
- `src0_stride` -指向 src0 的 stride 的指针
- `src1_stride` -指向 src1 的 stride 的指针
- `num_iter` -Newton 迭代次数

注意事项

- `dst`、`src0` 和 `src1` 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- `num_iter` 是性能参数，取值范围是 [1, 4]。

4.11.12 tpu_bdc_fp32_tunable_div_C

张量的元素除以常数。

```
void tpu_bdc_fp32_tunable_div_C(local_addr_t dst_addr, local_addr_t src_addr,
                                float C, const dim4 *shape, const dim4 *dst_stride,
                                const dim4 *src_stride, int num_iter)
```

$$\text{dst}(n, c, h, w) = \frac{\text{src}(n, c, h, w)}{C}$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `C` -常数
- `shape` -指向 dst 和 src 的 shape 的指针

- `dst_stride` - 指向 `dst` 的 `stride` 的指针
- `src_stride` - 指向 `src` 的 `stride` 的指针
- `num_iter` - Newton 迭代次数

注意事项

- `dst` 和 `src` 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 `[1, 65535]`。
- 如果张量的 `stride` 指针是 `NULL`，则此张量是 64-byte aligned layout，否则是 free layout。
- `num_iter` 是性能参数，取值范围是 `[1, 4]`。

4.11.13 tpu_bdc_fp32_tunable_C_div

常数除以张量的元素。

```
void tpu_bdc_fp32_tunable_C_div(local_addr_t dst_addr, local_addr_t src_addr,
                                float C, const dim4 *shape, const dim4 *dst_stride,
                                const dim4 *src_stride, int num_iter)
```

$$\text{dst}(n, c, h, w) = \frac{C}{\text{src}(n, c, h, w)}$$

参数

- `dst_addr` - `dst` 的地址
- `src_addr` - `src` 的地址
- `C` - 常数
- `shape` - 指向 `dst` 和 `src` 的 `shape` 的指针
- `dst_stride` - 指向 `dst` 的 `stride` 的指针
- `src_stride` - 指向 `src` 的 `stride` 的指针
- `num_iter` - Newton 迭代次数

注意事项

- `dst` 和 `src` 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 `[1, 65535]`。
- 如果张量的 `stride` 指针是 `NULL`，则此张量是 64-byte aligned layout，否则是 free layout。
- `num_iter` 是性能参数，取值范围是 `[1, 4]`。

4.11.14 tpu_bdc_fp32_mac

两个张量的元素相乘，再对结果做累加。

```
void tpu_bdc_fp32_mac(local_addr_t dst_addr, local_addr_t src0_addr, local_addr_t
                      src1_addr, const dim4 *shape, const dim4 *dst_stride, const
                      dim4 *src0_stride, const dim4 *src1_stride)
```

$$\text{dst}(n, c, h, w) = \text{dst}(n, c, h, w) + \text{src0}(n, c, h, w) \times \text{src1}(n, c, h, w)$$

参数

- `dst_addr` -dst 的地址
- `src0_addr` -src0 的地址
- `src1_addr` -src1 的地址
- `shape` -指向 dst、src0 和 src1 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针
- `src0_stride` -指向 src0 的 stride 的指针
- `src1_stride` -指向 src1 的 stride 的指针

注意事项

- dst、src0 和 src1 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。

4.11.15 tpu_bdc_fp32_mac_C

张量的元素和常数相乘，再对结果做累加。

```
void tpu_bdc_fp32_mac_C(local_addr_t dst_addr, local_addr_t src_addr, float C,
                        const dim4 *shape, const dim4 *dst_stride, const dim4
                        *src_stride)
```

$$\text{dst}(n, c, h, w) = \text{dst}(n, c, h, w) + \text{src}(n, c, h, w) \times C$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `C` -常数
- `shape` -指向 dst 和 src 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针
- `src_stride` -指向 src 的 stride 的指针

注意事项

- dst 和 src 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。

4.11.16 tpu_bdc_fp_diff_abs

计算两个张量的元素的差的绝对值。

```
void tpu_bdc_fp_diff_abs(local_addr_t dst_addr, local_addr_t src0_addr,
                        local_addr_t src1_addr, const dim4 *shape, const dim4
                        *dst_stride, const dim4 *src0_stride, const dim4 *src1_stride,
                        data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = |\text{src0}(n, c, h, w) - \text{src1}(n, c, h, w)|$$

参数

- `dst_addr` -dst 的地址
- `src0_addr` -src0 的地址
- `src1_addr` -src1 的地址
- `shape` -指向 dst、src0 和 src1 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针
- `src0_stride` -指向 src0 的 stride 的指针
- `src1_stride` -指向 src1 的 stride 的指针
- `dtype` -dst、src0 和 src1 的元素的数据类型

注意事项

- dst、src0 和 src1 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- `dtype` 的有效取值是 `DT_FP32`、`DT_FP16` 和 `DT_BFP16`。

4.11.17 tpu_bdc_fp_diff_abs_C

计算张量的元素与常数的差的绝对值。

```
void tpu_bdc_fp_diff_abs_C(local_addr_t dst_addr, local_addr_t src_addr, scalar_t
                           C, const dim4 *shape, const dim4 *dst_stride, const dim4
                           *src_stride, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = |\text{src}(n, c, h, w) - C|$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `C` -常数
- `shape` -指向 dst 和 src 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针
- `src_stride` -指向 src 的 stride 的指针
- `dtype` -dst 和 src 的元素和 C 的数据类型

注意事项

- dst 和 src 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- `dtype` 的有效取值是 `DT_FP32`、`DT_FP16` 和 `DT_BFP16`。

4.11.18 tpu_bdc_fp32_pow

计算以一个张量的元素为底，另一个张量的元素为指数的幂。

```
void tpu_bdc_fp32_pow(local_addr_t dst_addr, local_addr_t src0_addr, local_addr_t
                      src1_addr, local_addr_t work0_addr, local_addr_t
                      work1_addr, local_addr_t exp_coeff_addr, local_addr_t
                      log_coeff_addr, local_addr_t exp_table_addr, const dim4
                      *shape)
```

$$\text{dst}(n, c, h, w) = \text{src0}(n, c, h, w)^{\text{src1}(n, c, h, w)}$$

参数

- dst_addr -dst 的地址
- src0_addr -src0 的地址
- src1_addr -src1 的地址
- work0_addr -work0 的地址
- work1_addr -work1 的地址
- exp_coeff_addr -exp_coeff 的地址
- log_coeff_addr -log_coeff 的地址
- exp_table_addr -exp_table 的地址
- shape -指向 dst、src0、src1、work0 和 work1 的 shape 的指针

注意事项

- dst、src0、src1、work0 和 work1 从同一个 NPU 开始，都是 64-byte aligned layout。
- exp_coeff 通过 `tpu_bdc_load_fp32_exp_coeff()` 加载，log_coeff 通过 `tpu_bdc_load_fp32_log_coeff()` 加载，exp_table 通过 `tpu_bdc_load_fp32_exp_table()` 加载。
- work0 和 work1 存放中间结果，不允许 dst_addr、work0_addr 和 work1_addr 任意两个相等，允许 src0_addr 等于 dst_addr、work0_addr 或 work1_addr，此情况下 src0 的数据会被覆盖，允许 src1_addr 等于 dst_addr、work0_addr 或 work1_addr，此情况下 src1 的数据会被覆盖，允许 src0_addr 等于 src1_addr，如果 dst、work1 和 table 两两之间没有 bank conflicting，则性能更优。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]，shape->h * shape->w 小于等于 65535。
- src0 的元素的取值范围是 $(0, +\infty)$ ，如果 $\text{src1}(n, c, h, w) \times \log(\text{src0}(n, c, h, w))$ 小于 -103 或大于 88，则 $\text{dst}(n, c, h, w)$ 分别是 e^{-103} 和 e^{88} 。

4.11.19 tpu_bdc_fp32_pow_C

计算以张量的元素为底，常数为指数的幂。

```
void tpu_bdc_fp32_pow_C(local_addr_t dst_addr, local_addr_t src_addr,
                        local_addr_t work0_addr, local_addr_t work1_addr,
                        local_addr_t exp_coeff_addr, local_addr_t
                        log_coeff_addr, local_addr_t exp_table_addr, float C,
                        const dim4 *shape)
```

$$\text{dst}(n, c, h, w) = \text{src}(n, c, h, w)^C$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `work0_addr` -work0 的地址
- `work1_addr` -work1 的地址
- `exp_coeff_addr` -exp_coeff 的地址
- `log_coeff_addr` -log_coeff 的地址
- `exp_table_addr` -exp_table 的地址
- `C` -常数
- `shape` -指向 `dst`、`src`、`work0` 和 `work1` 的 `shape` 的指针

注意事项

- `dst`、`src`、`work0` 和 `work1` 从同一个 NPU 开始，都是 64-byte aligned layout。
- `exp_coeff` 通过 `tpu_bdc_load_fp32_exp_coeff()` 加载，`log_coeff` 通过 `tpu_bdc_load_fp32_log_coeff()` 加载，`exp_table` 通过 `tpu_bdc_load_fp32_exp_table()` 加载。
- `work0` 和 `work1` 存放中间结果，不允许 `dst_addr`、`work0_addr` 和 `work1_addr` 任意两个相等，允许 `src_addr` 等于 `dst_addr`、`work0_addr` 或 `work1_addr`，此情况下 `src` 的数据会被覆盖，如果 `dst`、`work1` 和 `table` 两两之间没有 bank conflicting，则性能更优。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 $[1, 65535]$ ，`shape->h * shape->w` 小于等于 65535。
- `src` 的元素的取值范围是 $(0, +\infty)$ ，如果 $C \times \log(\text{src}(n, c, h, w))$ 小于 -103 或大于 88，则 `dst(n, c, h, w)` 分别是 e^{-103} 和 e^{88} 。

4.11.20 tpu_bdc_fp32_C_pow

计算以常数为底，张量的元素为指数的幂。

```
void tpu_bdc_fp32_pow_C(local_addr_t dst_addr, local_addr_t src_addr,
                        local_addr_t work0_addr, local_addr_t work1_addr,
                        local_addr_t exp_coeff_addr, local_addr_t
                        exp_table_addr, float C, const dim4 *shape)
```

$$\text{dst}(n, c, h, w) = C^{\text{src}(n, c, h, w)}$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `work0_addr` -work0 的地址
- `work1_addr` -work1 的地址
- `exp_coeff_addr` -exp_coeff 的地址
- `exp_table_addr` -exp_table 的地址
- `C` -常数
- `shape` -指向 `dst`、`src`、`work0` 和 `work1` 的 `shape` 的指针

注意事项

- dst、src、work0 和 work1 从同一个 NPU 开始，都是 64-byte aligned layout。
- exp_coeff 通过 `tpu_bdc_load_fp32_exp_coeff()` 加载，exp_table 通过 `tpu_bdc_load_fp32_exp_table()` 加载。
- work0 和 work1 存放中间结果，不允许 dst_addr、work0_addr 和 work1_addr 任意两个相等，允许 src_addr 等于 dst_addr、work0_addr 或 work1_addr，此情况下 src 的数据会被覆盖，如果 dst、work1 和 table 两两之间没有 bank conflicting，则性能更优。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]，shape->h * shape->w 小于等于 65535。
- C 的取值范围是 $(0, +\infty)$ ，如果 $\text{src}(n, c, h, w) \times \log(C)$ 小于 -103 或大于 88，则 $\text{dst}(n, c, h, w)$ 分别是 e^{-103} 和 e^{88} 。

4.11.21 tpu_bdc_fp_vc_add

两个向量的元素交叉相加。

```
void tpu_bdc_fp_vc_add(local_addr_t dst_addr, local_addr_t src0_addr,
                      local_addr_t src1_addr, int src0_len, int src1_len, int
                      src0_len_per_channel, int src1_len_per_channel,
                      data_type_t dtype)
```

$$\text{dst}(m, n) = \text{src0}(m) + \text{src1}(n)$$

参数

- dst_addr -dst 的地址
- src0_addr -src0 的地址
- src1_addr -src1 的地址
- src0_len -src0 的长度
- src1_len -src1 的长度
- src0_len_per_channel -src0 在每个 channel 的长度
- src1_len_per_channel -src1 在每个 channel 的长度
- dtype -dst、src0 和 src1 的元素的数据类型

注意事项

- dst 和 src1 从同一个 NPU 开始。
- src0_len 和 src1_len 的取值范围是 [1, 65535]。
- dst 是 matrix layout，dst 的行数是 src0_len，列数是 src1_len，src0 和 src1 是 vector layout。
- dtype 的有效取值是 DT_FP32、DT_FP16 和 DT_BFP16。

4.11.22 tpu_bdc_fp_vc_sub

两个向量的元素交叉相减。

```
void tpu_bdc_fp_vc_sub(local_addr_t dst_addr, local_addr_t src0_addr,
                      local_addr_t src1_addr, int src0_len, int src1_len, int
                      src0_len_per_channel, int src1_len_per_channel,
                      data_type_t dtype)
```

$$\text{dst}(m, n) = \text{src0}(m) - \text{src1}(n)$$

参数

- dst_addr -dst 的地址
- src0_addr -src0 的地址
- src1_addr -src1 的地址
- src0_len -src0 的长度
- src1_len -src1 的长度
- src0_len_per_channel -src0 在每个 channel 的长度
- src1_len_per_channel -src1 在每个 channel 的长度
- dtype -dst、src0 和 src1 的元素的数据类型

注意事项

- dst 和 src1 从同一个 NPU 开始。
- src0_len 和 src1_len 的取值范围是 [1, 65535]。
- dst 是 matrix layout, dst 的行数是 src0_len, 列数是 src1_len, src0 和 src1 是 vector layout。
- dtype 的有效取值是DT_FP32、DT_FP16 和DT_BFP16。

4.11.23 tpu_bdc_fp_vc_mul

两个向量的元素交叉相乘。

```
void tpu_bdc_fp_vc_mul(local_addr_t dst_addr, local_addr_t src0_addr,
                      local_addr_t src1_addr, int src0_len, int src1_len, int
                      src0_len_per_channel, int src1_len_per_channel,
                      data_type_t dtype)
```

$$\text{dst}(m, n) = \text{src0}(m) \times \text{src1}(n)$$

参数

- dst_addr -dst 的地址
- src0_addr -src0 的地址
- src1_addr -src1 的地址
- src0_len -src0 的长度
- src1_len -src1 的长度
- src0_len_per_channel -src0 在每个 channel 的长度
- src1_len_per_channel -src1 在每个 channel 的长度

- `dtype -dst`、`src0` 和 `src1` 的元素的数据类型

注意事项

- `dst` 和 `src1` 从同一个 NPU 开始。
- `src0_len` 和 `src1_len` 的取值范围是 [1, 65535]。
- `dst` 是 matrix layout, `dst` 的行数是 `src0_len`, 列数是 `src1_len`, `src0` 和 `src1` 是 vector layout。
- `dtype` 的有效取值是 `DT_FP32`、`DT_FP16` 和 `DT_BFP16`。

4.11.24 tpu_bdc_fp32_vc_div

两个向量的元素交叉相除。

```
void tpu_bdc_fp32_vc_div(local_addr_t dst_addr, local_addr_t src0_addr,
                        local_addr_t src1_addr, int src0_len, int src1_len, int
                        src0_len_per_channel, int src1_len_per_channel)
```

$$dst(m, n) = \frac{src0(m)}{src1(n)}$$

参数

- `dst_addr` - `dst` 的地址
- `src0_addr` - `src0` 的地址
- `src1_addr` - `src1` 的地址
- `src0_len` - `src0` 的长度
- `src1_len` - `src1` 的长度
- `src0_len_per_channel` - `src0` 在每个 channel 的长度
- `src1_len_per_channel` - `src1` 在每个 channel 的长度

注意事项

- `dst` 和 `src1` 从同一个 NPU 开始。
- `src0_len` 和 `src1_len` 的取值范围是 [1, 65535]。
- `dst` 是 matrix layout, `dst` 的行数是 `src0_len`, 列数是 `src1_len`, `src0` 和 `src1` 是 vector layout。

4.12 整型二元操作

4.12.1 tpu_bdc_int_add

两个张量的元素相加，对结果做算数移位，再对结果做 saturation（可选）。

```
void tpu_bdc_int_add(local_addr_t dst_addr, local_addr_t src0_addr, local_addr_t
                    src1_addr, const dim4 *shape, const dim4 *dst_stride, const dim4
                    *src0_stride, const dim4 *src1_stride, data_type_t dst_dtype,
                    data_type_t src0_dtype, data_type_t src1_dtype, char shift,
                    rounding_mode_t rounding_mode, bool saturation)
```

$$dst(n, c, h, w) = \begin{cases} (src0(n, c, h, w) + src1(n, c, h, w)) \text{ 左移 } shift & \text{如果 } shift > 0 \\ (src0(n, c, h, w) + src1(n, c, h, w)) \text{ 右移 } -shift & \text{其他情况} \end{cases}$$

参数

- `dst_addr` -dst 的地址
- `src0_addr` -src0 的地址
- `src1_addr` -src1 的地址
- `shape` -指向 dst、src0 和 src1 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针
- `src0_stride` -指向 src0 的 stride 的指针
- `src1_stride` -指向 src1 的 stride 的指针
- `dst_dtype` -dst 的元素的数据类型
- `src0_dtype` -src0 的元素的数据类型
- `src1_dtype` -src1 的元素的数据类型
- `shift` -移位数
- `rounding_mode` -右移舍入模式
- `saturation` -饱和处理标志

注意事项

- dst、src0 和 src1 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- `shift` 的取值范围是 [-32, 31]。
- `dst_dtype`、`src0_dtype` 和 `src1_dtype` 的有效取值是 `DT_INT32`、`DT_UINT32`、`DT_INT16`、`DT_UINT16`、`DT_INT8` 和 `DT_UINT8`，`dst_dtype` 是无符号的当且仅当 `src0_dtype` 和 `src1_dtype` 都是无符号的。

4.12.2 tpu_bdc_int_add_C

张量的元素和常数相加，对结果做算数移位，再对结果做 saturation（可选）。

```
void tpu_bdc_int_add_C(local_addr_t dst_addr, local_addr_t src_addr, scalar_t C,
                      const dim4 *shape, const dim4 *dst_stride, const dim4
                      *src_stride, data_type_t dst_dtype, data_type_t src_dtype,
                      data_type_t C_dtype, char shift, rounding_mode_t
                      rounding_mode, bool saturation)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (\text{src}(n, c, h, w) + C) \text{ 左移 } \text{shift} & \text{如果 } \text{shift} > 0 \\ (\text{src}(n, c, h, w) + C) \text{ 右移 } -\text{shift} & \text{其他情况} \end{cases}$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `C` -常数
- `shape` -指向 dst 和 src 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针
- `src_stride` -指向 src 的 stride 的指针

- `dst_dtype` -dst 的元素的数据类型
- `src_dtype` -src 的元素的数据类型
- `C_dtype` -C 的数据类型
- `shift` -移位数
- `rounding_mode` -右移舍入模式
- `saturation` -饱和处理标志

注意事项

- `dst` 和 `src` 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。
- 如果张量的 `stride` 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- `shift` 的取值范围是 [-32, 31]。
- `dst_dtype`、`src_dtype` 和 `C_dtype` 的有效取值是 `DT_INT32`、`DT_UINT32`、`DT_INT16`、`DT_UINT16`、`DT_INT8` 和 `DT_UINT8`，`dst_dtype` 是无符号的当且仅当 `src_dtype` 和 `C_dtype` 都是无符号的。

4.12.3 tpu_bdc_int_pcs_add

两个张量的元素相加，对结果按 channel 做算数移位，再对结果做 saturation（可选）。

```
void tpu_bdc_int_pcs_add(local_addr_t dst_addr, local_addr_t src0_addr,
                        local_addr_t src1_addr, local_addr_t shift_addr, const
                        dim4 *shape, const dim4 *dst_stride, const dim4
                        *src0_stride, const dim4 *src1_stride, data_type_t
                        dst_dtype, data_type_t src0_dtype, data_type_t
                        src1_dtype, rounding_mode_t rounding_mode, bool
                        saturation)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (\text{src0}(n, c, h, w) + \text{src1}(n, c, h, w)) \text{ 左移 } \text{shift}(0, c, 0, 0) & \text{如果 } \text{shift}(0, c, 0, 0) > 0 \\ (\text{src0}(n, c, h, w) + \text{src1}(n, c, h, w)) \text{ 右移 } -\text{shift}(0, c, 0, 0) & \text{其他情况} \end{cases}$$

参数

- `dst_addr` -dst 的地址
- `src0_addr` -src0 的地址
- `src1_addr` -src1 的地址
- `shift_addr` -shift 的地址
- `shape` -指向 `dst`、`src0` 和 `src1` 的 `shape` 的指针
- `dst_stride` -指向 `dst` 的 `stride` 的指针
- `src0_stride` -指向 `src0` 的 `stride` 的指针
- `src1_stride` -指向 `src1` 的 `stride` 的指针
- `dst_dtype` -dst 的元素的数据类型
- `src0_dtype` -src0 的元素的数据类型
- `src1_dtype` -src1 的元素的数据类型
- `rounding_mode` -右移舍入模式
- `saturation` -饱和处理标志

注意事项

- dst、src0、src1 和 shift 从同一个 NPU 开始。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- shift 的 shape 是 [1, shape->c, 1, 1]，compact layout，元素的数据类型是DT_INT8，取值范围是 [-32, 31]。
- dst_dtype、src0_dtype 和 src1_dtype 的有效取值是DT_INT32、DT_UINT32、DT_INT16、DT_UINT16、DT_INT8 和DT_UINT8，dst_dtype 是无符号的当且仅当 src0_dtype 和 src1_dtype 都是无符号的。

4.12.4 tpu_bdc_int_pcs_add_C

张量的元素和常数相加，对结果按 channel 做算数移位，再对结果做 saturation（可选）。

```
void tpu_bdc_int_pcs_add_C(local_addr_t dst_addr, local_addr_t src_addr,
                          local_addr_t shift_addr, scalar_t C, const dim4 *shape,
                          const dim4 *dst_stride, const dim4 *src_stride,
                          data_type_t dst_dtype, data_type_t src_dtype,
                          data_type_t C_dtype, rounding_mode_t
                          rounding_mode, bool saturation)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (\text{src}(n, c, h, w) + C) \text{ 左移 } \text{shift}(0, c, 0, 0) & \text{如果 } \text{shift}(0, c, 0, 0) > 0 \\ (\text{src}(n, c, h, w) + C) \text{ 右移 } -\text{shift}(0, c, 0, 0) & \text{其他情况} \end{cases}$$

参数

- dst_addr -dst 的地址
- src_addr -src 的地址
- shift_addr -shift 的地址
- C -常数
- shape -指向 dst 和 src 的 shape 的指针
- dst_stride -指向 dst 的 stride 的指针
- src_stride -指向 src 的 stride 的指针
- dst_dtype -dst 的元素的数据类型
- src_dtype -src 的元素的数据类型
- C_dtype -C 的数据类型
- rounding_mode -右移舍入模式
- saturation -饱和处理标志

注意事项

- dst、src 和 shift 从同一个 NPU 开始。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- shift 的 shape 是 [1, shape->c, 1, 1]，compact layout，元素的数据类型是DT_INT8，取值范围是 [-32, 31]。

- `dst_dtype`、`src_dtype` 和 `C_dtype` 的有效取值是 `DT_INT32`、`DT_UINT32`、`DT_INT16`、`DT_UINT16`、`DT_INT8` 和 `DT_UINT8`，`dst_dtype` 是无符号的当且仅当 `src_dtype` 和 `C_dtype` 都是无符号的。

4.12.5 tpu_bdc_int_sub

两个张量的元素相减，对结果做算数移位，再对结果做 saturation（可选）。

```
void tpu_bdc_int_sub(local_addr_t dst_addr, local_addr_t src0_addr, local_addr_t
                    src1_addr, const dim4 *shape, const dim4 *dst_stride, const dim4
                    *src0_stride, const dim4 *src1_stride, data_type_t dst_dtype,
                    data_type_t src0_dtype, data_type_t src1_dtype, char shift,
                    rounding_mode_t rounding_mode, bool saturation)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (\text{src0}(n, c, h, w) - \text{src1}(n, c, h, w)) \text{ 左移 } \text{shift} & \text{如果 } \text{shift} > 0 \\ (\text{src0}(n, c, h, w) - \text{src1}(n, c, h, w)) \text{ 右移 } -\text{shift} & \text{其他情况} \end{cases}$$

参数

- `dst_addr` -dst 的地址
- `src0_addr` -src0 的地址
- `src1_addr` -src1 的地址
- `shape` -指向 dst、src0 和 src1 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针
- `src0_stride` -指向 src0 的 stride 的指针
- `src1_stride` -指向 src1 的 stride 的指针
- `dst_dtype` -dst 的元素的数据类型
- `src0_dtype` -src0 的元素的数据类型
- `src1_dtype` -src1 的元素的数据类型
- `shift` -移位数
- `rounding_mode` -右移舍入模式
- `saturation` -饱和处理标志

注意事项

- `dst`、`src0` 和 `src1` 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- `shift` 的取值范围是 [-32, 31]。
- `src0_dtype` 和 `src1_dtype` 的有效取值是 `DT_INT32`、`DT_UINT32`、`DT_INT16`、`DT_UINT16`、`DT_INT8` 和 `DT_UINT8`，`dst_dtype` 的有效取值是 `DT_INT32`、`DT_INT16` 和 `DT_INT8`。

4.12.6 tpu_bdc_int_sub_C

张量的元素减常数，对结果做算数移位，再对结果做 saturation（可选）。

```
void tpu_bdc_int_sub_C(local_addr_t dst_addr, local_addr_t src_addr, scalar_t C,
                      const dim4 *shape, const dim4 *dst_stride, const dim4
                      *src_stride, data_type_t dst_dtype, data_type_t src_dtype,
                      data_type_t C_dtype, char shift, rounding_mode_t
                      rounding_mode, bool saturation)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (\text{src}(n, c, h, w) - C) \text{ 左移 } \text{shift} & \text{如果 } \text{shift} > 0 \\ (\text{src}(n, c, h, w) - C) \text{ 右移 } -\text{shift} & \text{其他情况} \end{cases}$$

参数

- dst_addr -dst 的地址
- src_addr -src 的地址
- C -常数
- shape -指向 dst 和 src 的 shape 的指针
- dst_stride -指向 dst 的 stride 的指针
- src_stride -指向 src 的 stride 的指针
- dst_dtype -dst 的元素的数据类型
- src_dtype -src 的元素的数据类型
- C_dtype -C 的数据类型
- shift -移位数
- rounding_mode -右移舍入模式
- saturation -饱和处理标志

注意事项

- dst 和 src 从同一个 NPU 开始。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- shift 的取值范围是 [-32, 31]。
- src_dtype 和 C_dtype 的有效取值是DT_INT32、DT_UINT32、DT_INT16、DT_UINT16、DT_INT8 和DT_UINT8，dst_dtype 的有效取值是DT_INT32、DT_INT16 和DT_INT8。

4.12.7 tpu_bdc_int_C_sub

常数减张量的元素，对结果做算数移位，再对结果做 saturation（可选）。

```
void tpu_bdc_int_C_sub(local_addr_t dst_addr, local_addr_t src_addr, scalar_t C,
                      const dim4 *shape, const dim4 *dst_stride, const dim4
                      *src_stride, data_type_t dst_dtype, data_type_t src_dtype,
                      data_type_t C_dtype, char shift, rounding_mode_t
                      rounding_mode, bool saturation)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (C - \text{src}(n, c, h, w)) \text{ 左移 } \text{shift} & \text{如果 } \text{shift} > 0 \\ (C - \text{src}(n, c, h, w)) \text{ 右移 } -\text{shift} & \text{其他情况} \end{cases}$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `C` -常数
- `shape` -指向 dst 和 src 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针
- `src_stride` -指向 src 的 stride 的指针
- `dst_dtype` -dst 的元素的数据类型
- `src_dtype` -src 的元素的数据类型
- `C_dtype` -C 的数据类型
- `shift` -移位数
- `rounding_mode` -右移舍入模式
- `saturation` -饱和处理标志

注意事项

- dst 和 src 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- `shift` 的取值范围是 [-32, 31]。
- `src_dtype` 和 `C_dtype` 的有效取值是 `DT_INT32`、`DT_UINT32`、`DT_INT16`、`DT_UINT16`、`DT_INT8` 和 `DT_UINT8`，`dst_dtype` 的有效取值是 `DT_INT32`、`DT_INT16` 和 `DT_INT8`。

4.12.8 tpu_bdc_int_pcs_sub

两个张量的元素相减，对结果按 channel 做算数移位，再对结果做 saturation（可选）。

```
void tpu_bdc_int_pcs_sub(local_addr_t dst_addr, local_addr_t src0_addr,
                        local_addr_t src1_addr, local_addr_t shift_addr, const
                        dim4 *shape, const dim4 *dst_stride, const dim4
                        *src0_stride, const dim4 *src1_stride, data_type_t
                        dst_dtype, data_type_t src0_dtype, data_type_t
                        src1_dtype, rounding_mode_t rounding_mode, bool
                        saturation)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (\text{src0}(n, c, h, w) - \text{src1}(n, c, h, w)) \text{ 左移 } \text{shift}(0, c, 0, 0) & \text{如果 } \text{shift}(0, c, 0, 0) > 0 \\ (\text{src0}(n, c, h, w) - \text{src1}(n, c, h, w)) \text{ 右移 } -\text{shift}(0, c, 0, 0) & \text{其他情况} \end{cases}$$

参数

- `dst_addr` -dst 的地址
- `src0_addr` -src0 的地址
- `src1_addr` -src1 的地址
- `shift_addr` -shift 的地址
- `shape` -指向 dst、src0 和 src1 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针

- `src0_stride` –指向 `src0` 的 `stride` 的指针
- `src1_stride` –指向 `src1` 的 `stride` 的指针
- `dst_dtype` –`dst` 的元素的数据类型
- `src0_dtype` –`src0` 的元素的数据类型
- `src1_dtype` –`src1` 的元素的数据类型
- `rounding_mode` –右移舍入模式
- `saturation` –饱和处理标志

注意事项

- `dst`、`src0`、`src1` 和 `shift` 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。
- 如果张量的 `stride` 指针是 `NULL`，则此张量是 64-byte aligned layout，否则是 free layout。
- `shift` 的 `shape` 是 [1, `shape->c`, 1, 1]，compact layout，元素的数据类型是 `DT_INT8`，取值范围是 [-32, 31]。
- `src0_dtype` 和 `src1_dtype` 的有效取值是 `DT_INT32`、`DT_UINT32`、`DT_INT16`、`DT_UINT16`、`DT_INT8` 和 `DT_UINT8`，`dst_dtype` 的有效取值是 `DT_INT32`、`DT_INT16` 和 `DT_INT8`。

4.12.9 tpu_bdc_int_pcs_sub_C

张量的元素减常数，对结果按 channel 做算数移位，再对结果做 saturation（可选）。

```
void tpu_bdc_int_pcs_sub_C(local_addr_t dst_addr, local_addr_t src_addr,
                           local_addr_t shift_addr, scalar_t C, const dim4 *shape,
                           const dim4 *dst_stride, const dim4 *src_stride,
                           data_type_t dst_dtype, data_type_t src_dtype,
                           data_type_t C_dtype, rounding_mode_t
                           rounding_mode, bool saturation)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (\text{src}(n, c, h, w) - C) \text{ 左移 } \text{shift}(0, c, 0, 0) & \text{如果 } \text{shift}(0, c, 0, 0) > 0 \\ (\text{src}(n, c, h, w) - C) \text{ 右移 } -\text{shift}(0, c, 0, 0) & \text{其他情况} \end{cases}$$

参数

- `dst_addr` –`dst` 的地址
- `src_addr` –`src` 的地址
- `shift_addr` –`shift` 的地址
- `C` –常数
- `shape` –指向 `dst` 和 `src` 的 `shape` 的指针
- `dst_stride` –指向 `dst` 的 `stride` 的指针
- `src_stride` –指向 `src` 的 `stride` 的指针
- `dst_dtype` –`dst` 的元素的数据类型
- `src_dtype` –`src` 的元素的数据类型
- `C_dtype` –`C` 的数据类型
- `rounding_mode` –右移舍入模式
- `saturation` –饱和处理标志

Remarks

- dst、src 和 shift 从同一个 NPU 开始。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- shift 的 shape 是 [1, shape->c, 1, 1]，compact layout，元素的数据类型是DT_INT8，取值范围是 [-32, 31]。
- src_dtype 和 C_dtype 的有效取值是DT_INT32、DT_UINT32、DT_INT16、DT_UINT16、DT_INT8 和DT_UINT8，dst_dtype 的有效取值是DT_INT32、DT_INT16 和DT_INT8。

4.12.10 tpu_bdc_int_pcs_C_sub

常数减张量的元素，对结果按 channel 做算数移位，再对结果做 saturation（可选）。

```
void tpu_bdc_int_pcs_C_sub(local_addr_t dst_addr, local_addr_t src_addr,
                          local_addr_t shift_addr, scalar_t C, const dim4 *shape,
                          const dim4 *dst_stride, const dim4 *src_stride,
                          data_type_t dst_dtype, data_type_t src_dtype,
                          data_type_t C_dtype, rounding_mode_t
                          rounding_mode, bool saturation)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (C - \text{src}(n, c, h, w)) \text{ 左移 } \text{shift}(0, c, 0, 0) & \text{如果 } \text{shift}(0, c, 0, 0) > 0 \\ (C - \text{src}(n, c, h, w)) \text{ 右移 } -\text{shift}(0, c, 0, 0) & \text{其他情况} \end{cases}$$

参数

- dst_addr -dst 的地址
- src_addr -src 的地址
- shift_addr -shift 的地址
- C -常数
- shape -指向 dst 和 src 的 shape 的指针
- dst_stride -指向 dst 的 stride 的指针
- src_stride -指向 src 的 stride 的指针
- dst_dtype -dst 的元素的数据类型
- src_dtype -src 的元素的数据类型
- C_dtype -C 的数据类型
- rounding_mode -右移舍入模式
- saturation -饱和处理标志

Remarks

- dst、src 和 shift 从同一个 NPU 开始。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- shift 的 shape 是 [1, shape->c, 1, 1]，compact layout，元素的数据类型是DT_INT8，取值范围是 [-32, 31]。

- `src_dtype` 和 `C_dtype` 的有效取值是 `DT_INT32`、`DT_UINT32`、`DT_INT16`、`DT_UINT16`、`DT_INT8` 和 `DT_UINT8`，`dst_dtype` 的有效取值是 `DT_INT32`、`DT_INT16` 和 `DT_INT8`。

4.12.11 tpu_bdc_int_mul

两个张量的元素相乘，对结果做算数移位，再对结果做 saturation（可选）。

```
void tpu_bdc_int_mul(local_addr_t dst_addr, local_addr_t src0_addr, local_addr_t
    src1_addr, const dim4 *shape, const dim4 *dst_stride, const dim4
    *src0_stride, const dim4 *src1_stride, data_type_t dst_dtype,
    data_type_t src0_dtype, data_type_t src1_dtype, char shift,
    rounding_mode_t rounding_mode, bool saturation)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (\text{src0}(n, c, h, w) \times \text{src1}(n, c, h, w)) \text{ 左移 } \text{shift} & \text{如果 } \text{shift} > 0 \\ (\text{src0}(n, c, h, w) \times \text{src1}(n, c, h, w)) \text{ 右移 } -\text{shift} & \text{其他情况} \end{cases}$$

参数

- `dst_addr` -dst 的地址
- `src0_addr` -src0 的地址
- `src1_addr` -src1 的地址
- `shape` -指向 dst、src0 和 src1 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针
- `src0_stride` -指向 src0 的 stride 的指针
- `src1_stride` -指向 src1 的 stride 的指针
- `dst_dtype` -dst 的元素的数据类型
- `src0_dtype` -src0 的元素的数据类型
- `src1_dtype` -src1 的元素的数据类型
- `shift` -移位数
- `rounding_mode` -右移舍入模式
- `saturation` -饱和处理标志

注意事项

- dst、src0 和 src1 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- `shift` 的取值范围是 [-64, 31]。
- `dst_dtype`、`src0_dtype` 和 `src1_dtype` 的有效取值是 `DT_INT32`、`DT_UINT32`、`DT_INT16`、`DT_UINT16`、`DT_INT8` 和 `DT_UINT8`，`dst_dtype` 是无符号的当且仅当 `src0_dtype` 和 `src1_dtype` 都是无符号的。

4.12.12 tpu_bdc_int_mul_C

张量的元素和常数相乘，对结果做算数移位，再对结果做 saturation（可选）。

```
void tpu_bdc_int_mul_C(local_addr_t dst_addr, local_addr_t src_addr, scalar_t C,
                      const dim4 *shape, const dim4 *dst_stride, const dim4
                      *src_stride, data_type_t dst_dtype, data_type_t src_dtype,
                      data_type_t C_dtype, char shift, rounding_mode_t
                      rounding_mode, bool saturation)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (\text{src}(n, c, h, w) \times C) \text{ 左移 } \text{shift} & \text{如果 } \text{shift} > 0 \\ (\text{src}(n, c, h, w) \times C) \text{ 右移 } -\text{shift} & \text{其他情况} \end{cases}$$

参数

- dst_addr -dst 的地址
- src_addr -src 的地址
- C -常数
- shape -指向 dst 和 src 的 shape 的指针
- dst_stride -指向 dst 的 stride 的指针
- src_stride -指向 src 的 stride 的指针
- dst_dtype -dst 的元素的数据类型
- src_dtype -src 的元素的数据类型
- C_dtype -C 的数据类型
- shift -移位数
- rounding_mode -右移舍入模式
- saturation -饱和处理标志

注意事项

- dst 和 src 从同一个 NPU 开始。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- shift 的取值范围是 [-64, 31]。
- dst_dtype、src_dtype 和 C_dtype 的有效取值是 DT_INT32、DT_UINT32、DT_INT16、DT_UINT16、DT_INT8 和 DT_UINT8，dst_dtype 是无符号的当且仅当 src_dtype 和 C_dtype 都是无符号的。

4.12.13 tpu_bdc_int_pcs_mul

两个张量的元素相乘，对结果按 channel 做算数移位，再对结果做 saturation（可选）。

```
void tpu_bdc_int_pcs_mul(local_addr_t dst_addr, local_addr_t src0_addr,
                        local_addr_t src1_addr, local_addr_t shift_addr, const
                        dim4 *shape, const dim4 *dst_stride, const dim4
                        *src0_stride, const dim4 *src1_stride, data_type_t
                        dst_dtype, data_type_t src0_dtype, data_type_t
                        src1_dtype, rounding_mode_t rounding_mode, bool
                        saturation)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (\text{src0}(n, c, h, w) \times \text{src1}(n, c, h, w)) \text{ 左移 } \text{shift}(0, c, 0, 0) & \text{如果 } \text{shift}(0, c, 0, 0) > 0 \\ (\text{src0}(n, c, h, w) \times \text{src1}(n, c, h, w)) \text{ 右移 } -\text{shift}(0, c, 0, 0) & \text{其他情况} \end{cases}$$

参数

- `dst_addr` -dst 的地址
- `src0_addr` -src0 的地址
- `src1_addr` -src1 的地址
- `shift_addr` -shift 的地址
- `shape` -指向 dst、src0 和 src1 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针
- `src0_stride` -指向 src0 的 stride 的指针
- `src1_stride` -指向 src1 的 stride 的指针
- `dst_dtype` -dst 的元素的数据类型
- `src0_dtype` -src0 的元素的数据类型
- `src1_dtype` -src1 的元素的数据类型
- `rounding_mode` -右移舍入模式
- `saturation` -饱和处理标志

注意事项

- dst、src0、src1 和 shift 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- shift 的 shape 是 [1, `shape->c`, 1, 1]，compact layout，元素的数据类型是 `DT_INT8`，取值范围是 [-64, 31]。
- `dst_dtype`、`src0_dtype` 和 `src1_dtype` 的有效取值是 `DT_INT32`、`DT_UINT32`、`DT_INT16`、`DT_UINT16`、`DT_INT8` 和 `DT_UINT8`，`dst_dtype` 是无符号的当且仅当 `src0_dtype` 和 `src1_dtype` 都是无符号的。

4.12.14 tpu_bdc_int_pcs_mul_C

张量的元素和常数相乘，对结果按 channel 做算数移位，再对结果做 saturation（可选）。

```
void tpu_bdc_int_pcs_mul_C(local_addr_t dst_addr, local_addr_t src_addr,
                           local_addr_t shift_addr, scalar_t C, const dim4 *shape,
                           const dim4 *dst_stride, const dim4 *src_stride,
                           data_type_t dst_dtype, data_type_t src_dtype,
                           data_type_t C_dtype, rounding_mode_t
                           rounding_mode, bool saturation)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (\text{src}(n, c, h, w) \times C) \text{ 左移 } \text{shift}(0, c, 0, 0) & \text{如果 } \text{shift}(0, c, 0, 0) > 0 \\ (\text{src}(n, c, h, w) \times C) \text{ 右移 } -\text{shift}(0, c, 0, 0) & \text{其他情况} \end{cases}$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `shift_addr` -shift 的地址
- `C` -常数
- `shape` -指向 dst 和 src 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针
- `src_stride` -指向 src 的 stride 的指针
- `dst_dtype` -dst 的元素的数据类型
- `src_dtype` -src 的元素的数据类型
- `C_dtype` -C 的数据类型
- `rounding_mode` -右移舍入模式
- `saturation` -饱和处理标志

注意事项

- `dst`、`src` 和 `shift` 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- `shift` 的 shape 是 [1, `shape->c`, 1, 1]，compact layout，元素的数据类型是 `DT_INT8`，取值范围是 [-64, 31]。
- `dst_dtype`、`src_dtype` 和 `C_dtype` 的有效取值是 `DT_INT32`、`DT_UINT32`、`DT_INT16`、`DT_UINT16`、`DT_INT8` 和 `DT_UINT8`，`dst_dtype` 是无符号的当且仅当 `src_dtype` 和 `C_dtype` 都是无符号的。

4.12.15 tpu_bdc_int8_mac

两个张量的元素相乘，再对结果做累加，在累加之前会对结果的原数据算数左移，在累加之后对结果算数右移。

```
void tpu_bdc_int8_mac(local_addr_t dst_addr, local_addr_t src0_addr, local_addr_t
                      src1_addr, const dim4 *shape, const dim4 *dst_stride, const
                      dim4 *src0_stride, const dim4 *src1_stride, data_type_t
                      src0_dtype, data_type_t src1_dtype, unsigned char lshift,
                      unsigned char rshift, rounding_mode_t rounding_mode)
```

$$\text{dst}(n, c, h, w) = (\text{dst}(n, c, h, w) \text{ 左移 } \text{lshift} + \text{src0}(n, c, h, w) \times \text{src1}(n, c, h, w)) \text{ 右移 } \text{rshift}$$

参数

- `dst_addr` -dst 的地址
- `src0_addr` -src0 的地址
- `src1_addr` -src1 的地址
- `shape` -指向 dst、src0 和 src1 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针
- `src0_stride` -指向 src0 的 stride 的指针
- `src1_stride` -指向 src1 的 stride 的指针
- `src0_dtype` -src0 的元素的数据类型
- `src1_dtype` -src1 的元素的数据类型
- `lshift` -左移位数
- `rshift` -右移位数
- `rounding_mode` -右移舍入模式

注意事项

- dst、src0 和 src1 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- `lshift` 和 `rshift` 的取值范围是 [0, 31]。
- `src0_dtype` 和 `src1_dtype` 的有效取值是 `DT_INT8` and `DT_UINT8`，如果 `src0_dtype` 和 `src1_dtype` 都是 `DT_UINT8`，则 dst 的元素的数据类型是 `DT_UINT16`，累加前的元素的数据类型也被认定为 `DT_UINT16`，否则是 `DT_INT16`。

4.12.16 tpu_bdc_int8_mac_C

张量的元素与常数相乘，再对结果做累加，在累加之前会对结果的原数据算数左移，在累加之后对结果算数右移。

```
void tpu_bdc_int8_mac_C(local_addr_t dst_addr, local_addr_t src_addr, scalar_t C,
                        const dim4 *shape, const dim4 *dst_stride, const dim4
                        *src_stride, data_type_t src_dtype, data_type_t C_dtype,
                        unsigned char lshift, unsigned char rshift, rounding_mode_t
                        rounding_mode)
```

$$\text{dst}(n, c, h, w) = (\text{dst}(n, c, h, w) \text{ 左移 } lshift + \text{src}(n, c, h, w) \times C) \text{ 右移 } rshift$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `C` -常数
- `shape` -指向 dst 和 src 的 shape 的指针
- `dst_stride` -指向 dst 的 stride 的指针
- `src_stride` -指向 src 的 stride 的指针
- `src_dtype` -src0 的元素的数据类型

- `C_dtype` – `C` 的数据类型
- `lshift` – 左移位数
- `rshift` – 右移位数
- `rounding_mode` – 右移舍入模式

注意事项

- `dst` 和 `src` 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 `[1, 65535]`。
- 如果张量的 `stride` 指针是 `NULL`，则此张量是 64-byte aligned layout，否则是 free layout。
- `lshift` 和 `rshift` 的取值范围是 `[0, 31]`。
- `src_dtype` 和 `C_dtype` 的有效取值是 `DT_INT8` and `DT_UINT8`，如果 `src_dtype` 和 `C_dtype` 都是 `DT_UINT8`，则 `dst` 的元素的数据类型是 `DT_UINT16`，累加前的元素的数据类型也被认定为 `DT_UINT16`，否则是 `DT_INT16`。

4.12.17 tpu_bdc_int_min_C

两个张量的元素取最小值并对结果做算数移位。

```
void tpu_bdc_int_min_C(local_addr_t dst_addr, local_addr_t src_addr, scalar_t C,
                      const dim4 *shape, const dim4 *dst_stride, const dim4
                      *src_stride, data_type_t dtype, char shift, rounding_mode_t
                      rounding_mode)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \min(\text{src}(n, c, h, w), C) \text{ 左移 } \text{shift} & \text{如果 } \text{shift} > 0 \\ \min(\text{src}(n, c, h, w), C) \text{ 右移 } -\text{shift} & \text{其他情况} \end{cases}$$

参数

- `dst_addr` – `dst` 的地址
- `src_addr` – `src` 的地址
- `C` – 常数
- `shape` – 指向 `dst`、`src` 的 `shape` 的指针
- `dst_stride` – 指向 `dst` 的 `stride` 的指针
- `src_stride` – 指向 `src` 的 `stride` 的指针
- `dtype` – `dst` 的元素和 `src` 元素和常数 `C` 的数据类型
- `shift` – 移位数
- `rounding_mode` – 右移舍入模式

注意事项

- `dst` 和 `src` 从同一个 NPU 开始。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 `[1, 65535]`。
- 如果张量的 `stride` 指针是 `NULL`，则此张量是 64-byte aligned layout，否则是 free layout。
- `shift` 的取值范围是 `[-32, 31]`。
- `dtype` 的有效取值是 `DT_INT32`、`DT_UINT32`、`DT_INT16`、`DT_UINT16`、`DT_INT8` 和 `DT_UINT8`，

4.12.18 tpu_bdc_int_max_C

两个张量的元素取最大值并对结果做算数移位。

```
void tpu_bdc_int_max_C(local_addr_t dst_addr, local_addr_t src_addr, scalar_t C,
                       const dim4 *shape, const dim4 *dst_stride, const dim4
                       *src_stride, data_type_t dtype, char shift, rounding_mode_t
                       rounding_mode)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \max(\text{src}(n, c, h, w), C) \text{ 左移 } \text{shift} & \text{如果 } \text{shift} > 0 \\ \max(\text{src}(n, c, h, w), C) \text{ 右移 } -\text{shift} & \text{其他情况} \end{cases}$$

参数

- dst_addr -dst 的地址
- src_addr -src 的地址
- C -常数
- shape -指向 dst、src 的 shape 的指针
- dst_stride -指向 dst 的 stride 的指针
- src_stride -指向 src 的 stride 的指针
- dtype -dst 的元素和 src 元素和常数 C 的数据类型
- shift -移位数
- rounding_mode -右移舍入模式

注意事项

- dst 和 src 从同一个 NPU 开始。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。
- shift 的取值范围是 [-32, 31]。
- dtype 的有效取值是 DT_INT32、DT_UINT32、DT_INT16、DT_UINT16、DT_INT8 和 DT_UINT8，

4.12.19 tpu_bdc_int_vc_add

两个向量的元素交叉相加，对结果做 saturation（可选）

```
void tpu_bdc_int_vc_add(local_addr_t dst_addr, local_addr_t src0_addr,
                        local_addr_t src1_addr, int src0_len, int src1_len, int
                        src0_len_per_channel, int src1_len_per_channel,
                        data_type_t dst_dtype, data_type_t src0_dtype,
                        data_type_t src1_dtype, bool saturation)
```

$$\text{dst}(m, n) = \text{src0}(m) + \text{src1}(n)$$

参数

- dst_addr -dst 的地址
- src0_addr -src0 的地址
- src1_addr -src1 的地址

- `src0_len` - `src0` 的长度
- `src1_len` - `src1` 的长度
- `src0_len_per_channel` - `src0` 在每个 channel 的长度
- `src1_len_per_channel` - `src1` 在每个 channel 的长度
- `dst_dtype` - `dst` 的元素的数据类型
- `src0_dtype` - `src0` 的元素的数据类型
- `src1_dtype` - `src1` 的元素的数据类型
- `saturation` - 饱和处理标志

注意事项

- `dst` 和 `src1` 从同一个 NPU 开始。
- `src0_len` 和 `src1_len` 的取值范围是 [1, 65535]。
- `dst` 是 matrix layout, `dst` 的行数是 `src0_len`, 列数是 `src1_len`, `src0` 和 `src1` 是 vector layout。
- `dst_dtype`、`src0_dtype` 和 `src1_dtype` 的有效取值是 `DT_INT32`、`DT_UINT32`、`DT_INT16`、`DT_UINT16`、`DT_INT8` 和 `DT_UINT8`, `dst_dtype` 是无符号的当且仅当 `src0_dtype` 和 `src1_dtype` 都是无符号的。

4.12.20 tpu_bdc_int_vc_sub

两个向量的元素交叉相减, 对结果做 saturation (可选)

```
void tpu_bdc_int_vc_sub(local_addr_t dst_addr, local_addr_t src0_addr,
                        local_addr_t src1_addr, int src0_len, int src1_len, int
                        src0_len_per_channel, int src1_len_per_channel,
                        data_type_t dst_dtype, data_type_t src0_dtype,
                        data_type_t src1_dtype, bool saturation)
```

$$dst(m, n) = src0(m) - src1(n)$$

参数

- `dst_addr` - `dst` 的地址
- `src0_addr` - `src0` 的地址
- `src1_addr` - `src1` 的地址
- `src0_len` - `src0` 的长度
- `src1_len` - `src1` 的长度
- `src0_len_per_channel` - `src0` 在每个 channel 的长度
- `src1_len_per_channel` - `src1` 在每个 channel 的长度
- `dst_dtype` - `dst` 的元素的数据类型
- `src0_dtype` - `src0` 的元素的数据类型
- `src1_dtype` - `src1` 的元素的数据类型
- `saturation` - 饱和处理标志

注意事项

- `dst` 和 `src1` 从同一个 NPU 开始。
- `src0_len` 和 `src1_len` 的取值范围是 [1, 65535]。

- dst 是 matrix layout, dst 的行数是 src0_len, 列数是 src1_len, src0 和 src1 是 vector layout。
- dst_dtype、src0_dtype 和 src1_dtype 的有效取值是DT_INT32、DT_UINT32、DT_INT16、DT_UINT16、DT_INT8 和DT_UINT8, dst_dtype 的有效取值是DT_INT32、DT_INT16 和DT_INT8。

4.12.21 tpu_bdc_int_vc_mul

两个向量的元素交叉相乘, 对结果做 saturation (可选)

```
void tpu_bdc_int_vc_mul(local_addr_t dst_addr, local_addr_t src0_addr,
                        local_addr_t src1_addr, int src0_len, int src1_len, int
                        src0_len_per_channel, int src1_len_per_channel,
                        data_type_t dst_dtype, data_type_t src0_dtype,
                        data_type_t src1_dtype, bool saturation)
```

$$\text{dst}(m, n) = \text{src0}(m) \times \text{src1}(n)$$

参数

- dst_addr -dst 的地址
- src0_addr -src0 的地址
- src1_addr -src1 的地址
- src0_len -src0 的长度
- src1_len -src1 的长度
- src0_len_per_channel -src0 在每个 channel 的长度
- src1_len_per_channel -src1 在每个 channel 的长度
- dst_dtype -dst 的元素的数据类型
- src0_dtype -src0 的元素的数据类型
- src1_dtype -src1 的元素的数据类型
- saturation -饱和处理标志

注意事项

- dst 和 src1 从同一个 NPU 开始。
- src0_len 和 src1_len 的取值范围是 [1, 65535]。
- dst 是 matrix layout, dst 的行数是 src0_len, 列数是 src1_len, src0 和 src1 是 vector layout。
- dst_dtype、src0_dtype 和 src1_dtype 的有效取值是DT_INT32、DT_UINT32、DT_INT16、DT_UINT16、DT_INT8 和DT_UINT8, dst_dtype 是无符号的当且仅当 src0_dtype 和 src1_dtype 都是无符号的。

4.13 比较选择函数

4.13.1 tpu_bdc_greater_select

两个张量的元素比较大小，选取另外两个张量的元素作为结果。

```
void tpu_bdc_greater_select(local_addr_t dst_addr, const variable_t *src0, const
                           variable_t *src1, const variable_t *src2, const variable_t
                           *src3, const dim4 *shape, data_type_t src0_src1_dtype,
                           data_type_t dst_dtype)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{src2} & \text{如果 } \text{src0} > \text{src1} \\ \text{src3} & \text{其他情况} \end{cases}$$

对于上式中的 src0、src1、src2 和 src3，如果 src->type 是 **TENSOR**，则取 src(n, c, h, w)，如果 src->type 是 **VECTOR**，则取 src(0, c mod NPU_NUM, 0, 0)，如果 src->type 是 **SCALAR**，则取对应的常数。

参数

- dst_addr -dst 的地址
- src0 -指向 src0 的指针
- src1 -指向 src1 的指针
- src2 -指向 src2 的指针
- src3 -指向 src3 的指针
- shape -指向 dst 的 shape 的指针
- src0_src1_dtype -src0 和 src1 的元素的数据类型
- dst_dtype -dst、src2 和 src3 的元素的数据类型

注意事项

- dst 和 src 从同一个 NPU 开始，dst 是 64-byte aligned layout。
- 如果 src->type 是 **TENSOR**，则 src 与 dst 的 shape 相同，64-byte aligned layout，如果 src->type 是 **VECTOR**，则 src 的 shape 是 [1, shape->c, 1, 1]。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- src0_src1_dtype 的位宽小于等于 dst_dtype 的位宽。

4.13.2 tpu_bdc_less_select

两个张量的元素比较大小，选取另外两个张量的元素作为结果。

```
void tpu_bdc_less_select(local_addr_t dst_addr, const variable_t *src0, const
                         variable_t *src1, const variable_t *src2, const variable_t *src3,
                         const dim4 *shape, data_type_t src0_src1_dtype,
                         data_type_t dst_dtype)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{src2} & \text{如果 } \text{src0} < \text{src1} \\ \text{src3} & \text{其他情况} \end{cases}$$

对于上式中的 src0、src1、src2 和 src3，如果 src->type 是 **TENSOR**，则取 src(n, c, h, w)，如果 src->type 是 **VECTOR**，则取 src(0, c mod NPU_NUM, 0, 0)，如果 src->type 是 **SCALAR**，则取对应的常数。

参数

- `dst_addr` -dst 的地址
- `src0` -指向 `src0` 的指针
- `src1` -指向 `src1` 的指针
- `src2` -指向 `src2` 的指针
- `src3` -指向 `src3` 的指针
- `shape` -指向 `dst` 的 `shape` 的指针
- `src0_src1_dtype` -`src0` 和 `src1` 的元素的数据类型
- `dst_dtype` -`dst`、`src2` 和 `src3` 的元素的数据类型

注意事项

- `dst` 和 `src` 从同一个 NPU 开始, `dst` 是 64-byte aligned layout。
- 如果 `src->type` 是 **TENSOR**, 则 `src` 与 `dst` 的 `shape` 相同, 64-byte aligned layout, 如果 `src->type` 是 **VECTOR**, 则 `src` 的 `shape` 是 `[1, shape->c, 1, 1]`。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 `[1, 65535]`。
- `src0_src1_dtype` 的位宽小于等于 `dst_dtype` 的位宽。

4.13.3 tpu_bdc_equal_select

两个张量的元素比较大小, 选取另外两个张量的元素作为结果。

```
void tpu_bdc_equal_select(local_addr_t dst_addr, const variable_t *src0, const
                        variable_t *src1, const variable_t *src2, const variable_t
                        *src3, const dim4 *shape, data_type_t src0_src1_dtype,
                        data_type_t dst_dtype)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{src2} & \text{如果 } \text{src0} = \text{src1} \\ \text{src3} & \text{其他情况} \end{cases}$$

对于上式中的 `src0`、`src1`、`src2` 和 `src3`, 如果 `src->type` 是 **TENSOR**, 则取 `src(n, c, h, w)`, 如果 `src->type` 是 **VECTOR**, 则取 `src(0, c mod NPU_NUM, 0, 0)`, 如果 `src->type` 是 **SCALAR**, 则取对应的常数。

参数

- `dst_addr` -dst 的地址
- `src0` -指向 `src0` 的指针
- `src1` -指向 `src1` 的指针
- `src2` -指向 `src2` 的指针
- `src3` -指向 `src3` 的指针
- `shape` -指向 `dst` 的 `shape` 的指针
- `src0_src1_dtype` -`src0` 和 `src1` 的元素的数据类型
- `dst_dtype` -`dst`、`src2` 和 `src3` 的元素的数据类型

注意事项

- `dst` 和 `src` 从同一个 NPU 开始, `dst` 是 64-byte aligned layout。
- 如果 `src->type` 是 **TENSOR**, 则 `src` 与 `dst` 的 `shape` 相同, 64-byte aligned layout, 如果 `src->type` 是 **VECTOR**, 则 `src` 的 `shape` 是 `[1, shape->c, 1, 1]`。

- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 `[1, 65535]`。
- `src0_src1_dtype` 的位宽小于等于 `dst_dtype` 的位宽。

4.13.4 tpu_bdc_maximum_greater_select

两个张量的元素比较大小，选取其中较大的和另外两个张量的元素作为结果。

```
void tpu_bdc_maximum_greater_select(local_addr_t dst0_addr, local_addr_t
                                   dst1_addr, const variable_t *src0, const
                                   variable_t *src1, const variable_t *src2, const
                                   variable_t *src3, const dim4 *shape,
                                   data_type_t dst0_dtype, data_type_t
                                   dst1_dtype)
```

$$\text{dst0}(n, c, h, w) = \max(\text{src0}, \text{src1})$$

$$\text{dst1}(n, c, h, w) = \begin{cases} \text{src2} & \text{如果 } \text{src0} > \text{src1} \\ \text{src3} & \text{其他情况} \end{cases}$$

对于上式中的 `src0`、`src1`、`src2` 和 `src3`，如果 `src->type` 是 **TENSOR**，则取 `src(n, c, h, w)`，如果 `src->type` 是 **VECTOR**，则取 `src(0, c mod NPU_NUM, 0, 0)`，如果 `src->type` 是 **SCALAR**，则取对应的常数。

参数

- `dst0_addr` - `dst0` 的地址
- `dst1_addr` - `dst1` 的地址
- `src0` - 指向 `src0` 的指针
- `src1` - 指向 `src1` 的指针
- `src2` - 指向 `src2` 的指针
- `src3` - 指向 `src3` 的指针
- `shape` - 指向 `dst0` 和 `dst1` 的 `shape` 的指针
- `dst0_dtype` - `dst0`、`src0` 和 `src1` 的元素的数据类型
- `dst1_dtype` - `dst1`、`src2` 和 `src3` 的元素的数据类型

注意事项

- `dst` 和 `src` 从同一个 NPU 开始，`dst` 是 64-byte aligned layout。
- 如果 `src->type` 是 **TENSOR**，则 `src` 与 `dst` 的 `shape` 相同，64-byte aligned layout，如果 `src->type` 是 **VECTOR**，则 `src` 的 `shape` 是 `[1, shape->c, 1, 1]`。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 `[1, 65535]`。
- `dst0_dtype` 的位宽等于 `dst1_dtype` 的位宽。

4.13.5 tpu_bdc_minimum_less_select

两个张量的元素比较大小，选取其中较小的和另外两个张量的元素作为结果。

```
void tpu_bdc_minimum_less_select(local_addr_t dst0_addr, local_addr_t dst1_addr,
                                const variable_t *src0, const variable_t *src1, const
                                variable_t *src2, const variable_t *src3, const dim4
                                *shape, data_type_t dst0_dtype, data_type_t
                                dst1_dtype)
```

$$dst0(n, c, h, w) = \min(src0, src1)$$

$$dst1(n, c, h, w) = \begin{cases} src2 & \text{如果 } src0 < src1 \\ src3 & \text{其他情况} \end{cases}$$

对于上式中的 `src0`、`src1`、`src2` 和 `src3`，如果 `src->type` 是 **TENSOR**，则取 `src(n, c, h, w)`，如果 `src->type` 是 **VECTOR**，则取 `src(0, c mod NPU_NUM, 0, 0)`，如果 `src->type` 是 **SCALAR**，则取对应的常数。

参数

- `dst0_addr` - `dst0` 的地址
- `dst1_addr` - `dst1` 的地址
- `src0` - 指向 `src0` 的指针
- `src1` - 指向 `src1` 的指针
- `src2` - 指向 `src2` 的指针
- `src3` - 指向 `src3` 的指针
- `shape` - 指向 `dst0` 和 `dst1` 的 `shape` 的指针
- `dst0_dtype` - `dst0`、`src0` 和 `src1` 的元素的数据类型
- `dst1_dtype` - `dst1`、`src2` 和 `src3` 的元素的数据类型

注意事项

- `dst` 和 `src` 从同一个 NPU 开始，`dst` 是 64-byte aligned layout。
- 如果 `src->type` 是 **TENSOR**，则 `src` 与 `dst` 的 `shape` 相同，64-byte aligned layout，如果 `src->type` 是 **VECTOR**，则 `src` 的 `shape` 是 `[1, shape->c, 1, 1]`。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 `[1, 65535]`。
- `dst0_dtype` 的位宽等于 `dst1_dtype` 的位宽。

4.14 浮点矩阵操作

4.14.1 tpu_bdc_fp32_mm

两个矩阵相乘，结果按 column 加 bias（可选），再对结果做累加（可选）。

```
void tpu_bdc_fp32_mm(local_addr_t output_addr, local_addr_t left_addr,
                     local_addr_t right_addr, local_addr_t bias_addr, int
                     left_rows, int left_cols, int right_cols, int
                     left_cols_per_channel, int right_cols_per_channel, bool
                     has_bias, bool result_add)
```

$$output_{M \times N} = output_{M \times N} + left_{M \times K} \times right_{K \times N} + bias_{1 \times N}$$

$$\text{output}(m, n) = \text{output}(m, n) + \left(\sum_k \text{left}(m, k) \times \text{right}(k, n) \right) + \text{bias}(0, n)$$

参数

- `output_addr` - output 的地址
- `left_addr` - left 的地址
- `right_addr` - right 的地址
- `bias_addr` - bias 的地址
- `left_rows` - 左矩阵的行数
- `left_cols` - 左矩阵的列数
- `right_cols` - 右矩阵的列数
- `left_cols_per_channel` - 左矩阵在每个 channel 的列数
- `right_cols_per_channel` - 右矩阵在每个 channel 的列数
- `has_bias` - 加 bias 的标志
- `result_add` - 对结果做累加的标志

注意事项

- `output`、`right` 和 `bias` 从同一个 NPU 开始。
- `left_rows`、`left_cols` 和 `right_cols` 的取值范围是 [1, 65535]。
- `output`、`left`、`right` 和 `bias` 是 matrix layout, `output` 的行数是 `left_rows`, 列数是 `right_cols`, `right` 的行数是 `left_cols`, `bias` 的行数是 1, 列数是 `right_cols`。

4.14.2 tpu_bdc_fp32_mm_L_trans

左矩阵的转置乘以右矩阵, 结果按 column 加 bias (可选), 再对结果做累加 (可选)。

```
void tpu_bdc_fp32_mm_left_trans(local_addr_t output_addr, local_addr_t left_addr,
                                local_addr_t right_addr, local_addr_t bias_addr,
                                int left_rows, int left_cols, int right_cols, int
                                left_cols_per_channel, int right_cols_per_channel,
                                bool has_bias, bool result_add)
```

$$\text{output}_{M \times N} = \text{output}_{M \times N} + \text{left}_{K \times M}^T \times \text{right}_{K \times N} + \text{bias}_{1 \times N}$$

$$\text{output}(m, n) = \text{output}(m, n) + \left(\sum_k \text{left}(k, m) \times \text{right}(k, n) \right) + \text{bias}(0, n)$$

参数

- `output_addr` - output 的地址
- `left_addr` - left 的地址
- `right_addr` - right 的地址
- `bias_addr` - bias 的地址
- `left_rows` - 左矩阵的行数
- `left_cols` - 左矩阵的列数
- `right_cols` - 右矩阵的列数
- `left_cols_per_channel` - 左矩阵在每个 channel 的列数
- `right_cols_per_channel` - 右矩阵在每个 channel 的列数

- `has_bias` -加 `bias` 的标志
- `result_add` -对结果做累加的标志

注意事项

- `output`、`right` 和 `bias` 从同一个 NPU 开始。
- `left_rows`、`left_cols` 和 `right_cols` 的取值范围是 [1, 65535]。
- `output`、`left`、`right` 和 `bias` 是 matrix layout, `output` 的行数是 `left_cols`, 列数是 `right_cols`, `right` 的行数是 `left_rows`, `bias` 的行数是 1, 列数是 `right_cols`。

4.14.3 tpu_bdc_fp32_mm_L_const

两个矩阵相乘，其中左矩阵的所有元素都相同，结果按 column 加 `bias`（可选），再对结果做累加（可选）。

```
void tpu_bdc_fp32_mm_left_const(local_addr_t output_addr, local_addr_t
                                right_addr, local_addr_t bias_addr, float C, int
                                left_rows, int left_cols, int right_cols, int
                                right_cols_per_channel, bool has_bias, bool
                                result_add)
```

$$\text{output}_{M \times N} = \text{output}_{M \times N} + \text{left}_{M \times K} \times \text{right}_{K \times N} + \text{bias}_{1 \times N}$$

$$\text{output}(m, n) = \text{output}(m, n) + \left(\sum_k C \times \text{right}(k, n) \right) + \text{bias}(0, n)$$

参数

- `output_addr` -`output` 的地址
- `right_addr` -`right` 的地址
- `bias_addr` -`bias` 的地址
- `C` -左矩阵元素值
- `left_rows` -左矩阵的行数
- `left_cols` -左矩阵的列数
- `right_cols` -右矩阵的列数
- `right_cols_per_channel` -右矩阵在每个 channel 的列数
- `has_bias` -加 `bias` 的标志
- `result_add` -对结果做累加的标志

注意事项

- `output`、`right` 和 `bias` 从同一个 NPU 开始。
- `left_rows`、`left_cols` 和 `right_cols` 的取值范围是 [1, 65535]。
- `output`、`right` 和 `bias` 是 matrix layout, `output` 的行数是 `left_rows`, 列数是 `right_cols`, `right` 的行数是 `left_cols`, `bias` 的行数是 1, 列数是 `right_cols`。

4.14.4 tpu_bdc_fp_mm

两个矩阵相乘，再对结果做累加（可选）。

```
void tpu_bdc_fp_mm(local_addr_t output_addr, local_addr_t left_addr, local_addr_t
                  right_addr, int left_rows, int left_cols, int right_cols,
                  data_type_t output_dtype, data_type_t left_right_dtype, bool
                  result_add)
```

$$\text{output}_{M \times N} = \text{output}_{M \times N} + \text{left}_{M \times K} \times \text{right}_{K \times N}$$

$$\text{output}(m, n) = \text{output}(m, n) + \sum_k \text{left}(m, k) \times \text{right}(k, n)$$

参数

- output_addr - output 的地址
- left_addr - left 的地址
- right_addr - right 的地址
- left_rows - 左矩阵的行数
- left_cols - 左矩阵的列数
- right_cols - 右矩阵的列数
- output_dtype - output 的元素的数据类型
- left_right_dtype - left 和 right 的元素的数据类型
- result_add - 对结果做累加的标志

注意事项

- output、left 和 right 从 NPU 0 开始，都是 64-byte aligned layout。
- output 的 shape 是 [1, left_rows, 1, right_cols]，left 的 shape 是 [1, left_rows, 1, left_cols]，right 的 shape 是 [1, left_cols, 1, right_cols]。
- left_rows、left_cols 和 right_cols 的取值范围是 [1, 65535]。
- output_dtype 的有效取值是 DT_FP32 或与 left_right_dtype 相同，left_right_dtype 的有效取值是 DT_FP16 和 DT_BFP16。
- 如果对结果做累加，则累加前的 output 的元素的数据类型会被认定为 DT_FP32，而不是 DT_FP16 或 DT_BFP16。

4.14.5 tpu_bdc_fp_mm_R_trans

左矩阵乘以右矩阵的转置。

```
void tpu_bdc_fp_mm_R_trans(local_addr_t output_addr, local_addr_t left_addr,
                           local_addr_t right_addr, int left_rows, int left_cols, int
                           right_rows, data_type_t output_dtype, data_type_t
                           left_right_dtype)
```

$$\text{output}_{M \times N} = \text{left}_{M \times K} \times \text{right}_{N \times K}^T$$

$$\text{output}(m, n) = \sum_k \text{left}(m, k) \times \text{right}(n, k)$$

参数

- output_addr - output 的地址

- left_addr -left 的地址
- right_addr -right 的地址
- left_rows -左矩阵的行数
- left_cols -左矩阵的列数
- right_rows -右矩阵的行数
- output_dtype -output 的元素的数据类型
- left_right_dtype -left 和 right 的元素的数据类型

注意事项

- output、left 和 right 从 NPU 0 开始，都是 64-byte aligned layout。
- output 的 shape 是 [1, left_rows, 1, right_rows], left 的 shape 是 [1, left_rows, 1, left_cols], right 的 shape 是 [1, right_rows, 1, left_cols]。
- left_rows、left_cols 和 right_rows 的取值范围是 [1, 65535]。
- output_dtype 的有效取值是DT_FP32 或与 left_right_dtype 相同，left_right_dtype 的有效取值是DT_FP16 和DT_BFP16。

4.14.6 tpu_bdc_fp_mm_all_trans

两个矩阵的转置相乘，结果也转置，再对结果做累加（可选）。

```
void tpu_bdc_fp_mm_all_trans(local_addr_t output_addr, local_addr_t left_addr,
                             local_addr_t right_addr, int left_rows, int left_cols,
                             int right_rows, data_type_t output_dtype,
                             data_type_t left_right_dtype, bool result_add)
```

$$\text{output}_{N \times M}^T = \text{output}_{N \times M}^T + \text{left}_{K \times M}^T \times \text{right}_{N \times K}^T$$

$$\text{output}(n, m) = \text{output}(n, m) + \sum_k \text{left}(k, m) \times \text{right}(n, k)$$

参数

- output_addr -output 的地址
- left_addr -left 的地址
- right_addr -right 的地址
- left_rows -左矩阵的行数
- left_cols -左矩阵的列数
- right_rows -右矩阵的行数
- output_dtype -output 的元素的数据类型
- left_right_dtype -left 和 right 的元素的数据类型
- result_add -对结果做累加的标志

注意事项

- output、left 和 right 从 NPU 0 开始，都是 64-byte aligned layout。
- output 的 shape 是 [1, right_rows, 1, left_cols], left 的 shape 是 [1, left_rows, 1, left_cols], right 的 shape 是 [1, right_rows, 1, left_rows]。
- left_rows、left_cols 和 right_rows 的取值范围是 [1, 65535]。

- `output_dtype` 的有效取值是 `DT_FP32` 或与 `left_right_dtype` 相同, `left_right_dtype` 的有效取值是 `DT_FP16` 和 `DT_BFP16`。
- 如果对结果做累加, 则累加前的 `output` 的元素的数据类型会被认定为 `DT_FP32`, 而不是 `DT_FP16` 或 `DT_BFP16`。

4.14.7 tpu_bdc_fp_mm_L_const

两个矩阵相乘, 其中左矩阵的所有元素都相同, 再对结果做累加 (可选)。

```
void tpu_bdc_fp_mm_L_const(local_addr_t output_addr, local_addr_t right_addr,
                           scalar_t C, int left_rows, int left_cols, int right_cols,
                           data_type_t output_dtype, data_type_t
                           right_C_dtype, bool result_add)
```

$$\text{output}_{M \times N} = \text{output}_{M \times N} + \text{left}_{M \times K} \times \text{right}_{K \times N}$$

$$\text{output}(m, n) = \text{output}(m, n) + \sum_k C \times \text{right}(k, n)$$

参数

- `output_addr` - `output` 的地址
- `right_addr` - `right` 的地址
- `C` - 左矩阵元素值
- `left_rows` - 左矩阵的行数
- `left_cols` - 左矩阵的列数
- `right_cols` - 右矩阵的列数
- `output_dtype` - `output` 的元素的数据类型
- `right_C_dtype` - `right` 的元素和 `C` 的数据类型
- `result_add` - 对结果做累加的标志

注意事项

- `output` 和 `right` 从 NPU 0 开始, 都是 64-byte aligned layout。
- `output` 的 shape 是 `[1, left_rows, 1, right_cols]`, `left` 的 shape 是 `[1, left_rows, 1, left_cols]`, `right` 的 shape 是 `[1, left_cols, 1, right_cols]`。
- `left_rows`、`left_cols` 和 `right_cols` 的取值范围是 `[1, 65535]`。
- `output_dtype` 的有效取值是 `DT_FP32` 或与 `right_C_dtype` 相同, `right_C_dtype` 的有效取值是 `DT_FP16` 和 `DT_BFP16`。
- 如果对结果做累加, 则累加前的 `output` 的元素的数据类型会被认定为 `DT_FP32`, 而不是 `DT_FP16` 或 `DT_BFP16`。

4.14.8 tpu_bdc_fp_mm_R_const

两个矩阵相乘，其中右矩阵的所有元素都相同，再对结果做累加（可选）。

```
void tpu_bdc_fp_mm_R_const(local_addr_t output_addr, local_addr_t left_addr,
                           scalar_t C, int left_rows, int left_cols, int right_cols,
                           data_type_t output_dtype, data_type_t
                           left_C_dtype, bool result_add)
```

$$\text{output}_{M \times N} = \text{output}_{M \times N} + \text{left}_{M \times K} \times \text{right}_{K \times N}$$

$$\text{output}(m, n) = \text{output}(m, n) + \sum_k \text{left}(m, k) \times C$$

参数

- output_addr - output 的地址
- left_addr - left 的地址
- C - 右矩阵元素值
- left_rows - 左矩阵的行数
- left_cols - 左矩阵的列数
- right_cols - 右矩阵的列数
- output_dtype - output 的元素的数据类型
- left_C_dtype - left 的元素和 C 的数据类型
- result_add - 对结果做累加的标志

注意事项

- output 和 left 从 NPU 0 开始，都是 64-byte aligned layout。
- output 的 shape 是 [1, left_rows, 1, right_cols]，left 的 shape 是 [1, left_rows, 1, left_cols]，right 的 shape 是 [1, left_cols, 1, right_cols]。
- left_rows、left_cols 和 right_cols 的取值范围是 [1, 65535]。
- output_dtype 的有效取值是 DT_FP32 或与 left_C_dtype 相同，left_C_dtype 的有效取值是 DT_FP16 和 DT_BFP16。
- 如果对结果做累加，则累加前的 output 的元素的数据类型会被认定为 DT_FP32，而不是 DT_FP16 或 DT_BFP16。

4.14.9 tpu_bdc_fp_mm_L_const_R_trans

左矩阵乘以右矩阵的转置，其中左矩阵的所有元素都相同。

```
void tpu_bdc_fp_mm_L_const_R_trans(local_addr_t output_addr, local_addr_t
                                     right_addr, scalar_t C, int left_rows, int
                                     left_cols, int right_rows, data_type_t
                                     output_dtype, data_type_t right_C_dtype)
```

$$\text{output}_{M \times N} = \text{left}_{M \times K} \times \text{right}_{N \times K}^T$$

$$\text{output}(m, n) = \sum_k C \times \text{right}(n, k)$$

参数

- output_addr - output 的地址

- right_addr -right 的地址
- C -左矩阵元素值
- left_rows -左矩阵的行数
- left_cols -左矩阵的列数
- right_rows -右矩阵的行数
- output_dtype -output 的元素的数据类型
- right_C_dtype -right 的元素和 C 的数据类型

注意事项

- output 和 right 从 NPU 0 开始，都是 64-byte aligned layout。
- output 的 shape 是 [1, left_rows, 1, right_rows], left 的 shape 是 [1, left_rows, 1, left_cols], right 的 shape 是 [1, right_rows, 1, left_cols]。
- left_rows、left_cols 和 right_rows 的取值范围是 [1, 65535]。
- output_dtype 的有效取值是DT_FP32 或与 right_C_dtype 相同，right_C_dtype 的有效取值是DT_FP16 和DT_BFP16。

4.14.10 tpu_bdc_fp_mm_L_const_all_trans

两个矩阵的转置相乘，其中左矩阵的所有元素都相同，结果也转置，再对结果做累加（可选）。

```
void tpu_bdc_fp_mm_L_const_all_trans(local_addr_t output_addr, local_addr_t
                                     right_addr, scalar_t C, int left_rows, int
                                     left_cols, int right_rows, data_type_t
                                     output_dtype, data_type_t right_C_dtype,
                                     bool result_add)
```

$$\text{output}_{N \times M}^T = \text{output}_{N \times M}^T + \text{left}_{K \times M}^T \times \text{right}_{N \times K}^T$$

$$\text{output}(n, m) = \text{output}(n, m) + \sum_k C \times \text{right}(n, k)$$

参数

- output_addr -output 的地址
- right_addr -right 的地址
- C -左矩阵元素值
- left_rows -左矩阵的行数
- left_cols -左矩阵的列数
- right_rows -右矩阵的行数
- output_dtype -output 的元素的数据类型
- right_C_dtype -right 的元素和 C 的数据类型
- result_add -对结果做累加的标志

注意事项

- output 和 right 从 NPU 0 开始，都是 64-byte aligned layout。
- output 的 shape 是 [1, right_rows, 1, left_cols], left 的 shape 是 [1, left_rows, 1, left_cols], right 的 shape 是 [1, right_rows, 1, left_rows]。
- left_rows、left_cols 和 right_rows 的取值范围是 [1, 65535]。

- `output_dtype` 的有效取值是 `DT_FP32` 或与 `right_C_dtype` 相同, `right_C_dtype` 的有效取值是 `DT_FP16` 和 `DT_BFP16`。
- 如果对结果做累加, 则累加前的 `output` 的元素的数据类型会被认定为 `DT_FP32`, 而不是 `DT_FP16` 或 `DT_FP16`。

4.14.11 tpu_bdc_fp_mm_R_const_all_trans

两个矩阵的转置相乘, 其中右矩阵的所有元素都相同, 结果也转置, 再对结果做累加 (可选)。

```
void tpu_bdc_fp_mm_R_const_all_trans(local_addr_t output_addr, local_addr_t
                                     left_addr, scalar_t C, int left_rows, int
                                     left_cols, int right_rows, data_type_t
                                     output_dtype, data_type_t left_C_dtype,
                                     bool result_add)
```

$$\text{output}_{N \times M}^T = \text{output}_{N \times M}^T + \text{left}_{K \times M}^T \times \text{right}_{N \times K}^T$$

$$\text{output}(n, m) = \text{output}(n, m) + \sum_k \text{left}(k, m) \times C$$

参数

- `output_addr` - `output` 的地址
- `left_addr` - `left` 的地址
- `C` - 右矩阵元素值
- `left_rows` - 左矩阵的行数
- `left_cols` - 左矩阵的列数
- `right_rows` - 右矩阵的行数
- `output_dtype` - `output` 的元素的数据类型
- `left_C_dtype` - `left` 的元素和 `C` 的数据类型
- `result_add` - 对结果做累加的标志

注意事项

- `output` 和 `left` 从 NPU 0 开始, 都是 64-byte aligned layout。
- `output` 的 shape 是 `[1, right_rows, 1, left_cols]`, `left` 的 shape 是 `[1, left_rows, 1, left_cols]`, `right` 的 shape 是 `[1, right_rows, 1, left_rows]`。
- `left_rows`、`left_cols` 和 `right_rows` 的取值范围是 `[1, 65535]`。
- `output_dtype` 的有效取值是 `DT_FP32` 或与 `left_C_dtype` 相同, `left_C_dtype` 的有效取值是 `DT_FP16` 和 `DT_BFP16`。
- 如果对结果做累加, 则累加前的 `output` 的元素的数据类型会被认定为 `DT_FP32`, 而不是 `DT_FP16` 或 `DT_FP16`。

4.15 整型矩阵操作

4.15.1 tpu_bdc_int_mm

两个矩阵相乘，对结果做算术移位。

```
void tpu_bdc_int_mm(local_addr_t output_addr, local_addr_t left_addr,
                    local_addr_t right_addr, int left_rows, int left_cols, int
                    right_cols, int left_cols_per_channel, int
                    right_cols_per_channel, data_type_t left_dtype, data_type_t
                    right_dtype, char shift, rounding_mode_t rounding_mode)
```

$$\text{output}_{M \times N} = (\text{left}_{M \times K} \times \text{right}_{K \times N}) \text{ 右移 } - \text{shift}$$

$$\text{output}(m, n) = \left(\sum_k \text{left}(m, k) \times \text{right}(k, n) \right) \text{ 右移 } - \text{shift}$$

参数

- output_addr - output 的地址
- left_addr - left 的地址
- right_addr - right 的地址
- left_rows - 左矩阵的行数
- left_cols - 左矩阵的列数
- right_cols - 右矩阵的列数
- left_cols_per_channel - 左矩阵在每个 channel 的列数
- right_cols_per_channel - 右矩阵在每个 channel 的列数
- left_dtype - left 的元素的数据类型
- right_dtype - right 的元素的数据类型
- shift - 移位数
- rounding_mode - 右移舍入模式

注意事项

- output 和 right 从同一个 NPU 开始。
- left_rows、left_cols 和 right_cols 的取值范围是 [1, 65535]。
- output、left 和 right 是 matrix layout, output 的行数是 left_rows, 列数是 right_cols, right 的行数是 left_cols。
- left_dtype 和 right_dtype 的位宽相同，有效取值是 DT_INT32、DT_UINT32、DT_INT16、DT_UINT16、DT_INT8 和 DT_UINT8，如果 left_dtype 和 right_dtype 都是无符号的，则 output 的元素的数据类型是 DT_UINT32，否则是 DT_INT32。
- shift 的取值范围是 [-32, 0]，移位只有当 left_dtype 和 right_dtype 是 DT_INT32 或 DT_UINT32 时生效。

4.15.2 tpu_bdc_int_mm_L_trans

左矩阵的转置乘以右矩阵，对结果做算术移位。

```
void tpu_bdc_int_mm_L_trans(local_addr_t output_addr, local_addr_t left_addr,
                             local_addr_t right_addr, int left_rows, int left_cols,
                             int right_cols, int left_cols_per_channel, int
                             right_cols_per_channel, data_type_t left_dtype,
                             data_type_t right_dtype, char shift, rounding_mode_t
                             rounding_mode)
```

$$\text{output}_{M \times N} = (\text{left}_{K \times M}^T \times \text{right}_{K \times N}) \text{ 右移 } - \text{shift}$$

$$\text{output}(m, n) = \left(\sum_k \text{left}(k, m) \times \text{right}(k, n) \right) \text{ 右移 } - \text{shift}$$

参数

- output_addr - output 的地址
- left_addr - left 的地址
- right_addr - right 的地址
- left_rows - 左矩阵的行数
- left_cols - 左矩阵的列数
- right_cols - 右矩阵的列数
- left_cols_per_channel - 左矩阵在每个 channel 的列数
- right_cols_per_channel - 右矩阵在每个 channel 的列数
- left_dtype - left 的元素的数据类型
- right_dtype - right 的元素的数据类型
- shift - 移位数
- rounding_mode - 右移舍入模式

注意事项

- output 和 right 从同一个 NPU 开始。
- left_rows、left_cols 和 right_cols 的取值范围是 [1, 65535]。
- output、left 和 right 是 matrix layout，output 的行数是 left_cols，列数是 right_cols，right 的行数是 left_rows。
- left_dtype 和 right_dtype 的位宽相同，有效取值是 DT_INT32、DT_UINT32、DT_INT16、DT_UINT16、DT_INT8 和 DT_UINT8，如果 left_dtype 和 right_dtype 都是无符号的，则 output 的元素的数据类型是 DT_UINT32，否则是 DT_INT32。
- shift 的取值范围是 [-32, 0]，移位只有当 left_dtype 和 right_dtype 是 DT_INT32 或 DT_UINT32 时生效。

4.15.3 tpu_bdc_int_mm_L_const

两个矩阵相乘，其中左矩阵的所有元素都相同，对结果做算术移位。

```
void tpu_bdc_int_mm_L_const(local_addr_t output_addr, local_addr_t right_addr,
                           scalar_t C, int left_rows, int left_cols, int right_cols,
                           int right_cols_per_channel, data_type_t C_dtype,
                           data_type_t right_dtype, char shift,
                           rounding_mode_t rounding_mode)
```

$$\text{output}_{M \times N} = (\text{left}_{M \times K} \times \text{right}_{K \times N}) \text{ 右移 } - \text{shift}$$

$$\text{output}(m, n) = \left(\sum_k C \times \text{right}(k, n) \right) \text{ 右移 } - \text{shift}$$

参数

- output_addr - output 的地址
- right_addr - right 的地址
- C - 常数
- left_rows - 左矩阵的行数
- left_cols - 左矩阵的列数
- right_cols - 右矩阵的列数
- right_cols_per_channel - 右矩阵在每个 channel 的列数
- C_dtype - C 的数据类型
- right_dtype - right 的元素的数据类型
- shift - 移位数
- rounding_mode - 右移舍入模式

注意事项

- output 和 right 从同一个 NPU 开始。
- left_rows、left_cols 和 right_cols 的取值范围是 [1, 65535]。
- output 和 right 是 matrix layout, output 的行数是 left_rows, 列数是 right_cols, right 的行数是 left_cols。
- C_dtype 和 right_dtype 的位宽相同，有效取值是 DT_INT32、DT_UINT32、DT_INT16、DT_UINT16、DT_INT8 和 DT_UINT8，如果 C_dtype 和 right_dtype 都是无符号的，则 output 的元素的数据类型是 DT_UINT32，否则是 DT_INT32。
- shift 的取值范围是 [-32, 0]，移位只有当 C_dtype 和 right_dtype 是 DT_INT32 或 DT_UINT32 时生效。

4.15.4 tpu_bdc_int_pcs_mm

两个矩阵相乘，对结果按 column 做算术移位。

```
void tpu_bdc_int_pcs_mm(local_addr_t output_addr, local_addr_t left_addr,
                        local_addr_t right_addr, local_addr_t shift_addr, int
                        left_rows, int left_cols, int right_cols, int
                        left_cols_per_channel, int right_cols_per_channel,
                        data_type_t left_dtype, data_type_t right_dtype,
                        rounding_mode_t rounding_mode)
```

$$\text{output}_{M \times N} = (\text{left}_{M \times K} \times \text{right}_{K \times N}) \text{ 右移 } - \text{shift}_{1 \times N}$$

$$\text{output}(m, n) = \left(\sum_k \text{left}(m, k) \times \text{right}(k, n) \right) \text{ 右移 } - \text{shift}(0, n)$$

参数

- output_addr - output 的地址
- left_addr - left 的地址
- right_addr - right 的地址
- shift_addr - shift 的地址
- left_rows - 左矩阵的行数
- left_cols - 左矩阵的列数
- right_cols - 右矩阵的列数
- left_cols_per_channel - 左矩阵在每个 channel 的列数
- right_cols_per_channel - 右矩阵在每个 channel 的列数
- left_dtype - left 的元素的数据类型
- right_dtype - right 的元素的数据类型
- rounding_mode - 右移舍入模式

注意事项

- output、right 和 shift 从同一个 NPU 开始。
- left_rows、left_cols 和 right_cols 的取值范围是 [1, 65535]。
- output、left、right 和 shift 是 matrix layout, output 的行数是 left_rows, 列数是 right_cols, right 的行数是 left_cols, shift 的行数是 1, 列数是 right_cols。
- left_dtype 和 right_dtype 的位宽相同, 有效取值是 DT_INT32、DT_UINT32、DT_INT16、DT_UINT16、DT_INT8 和 DT_UINT8, 如果 left_dtype 和 right_dtype 都是无符号的, 则 output 的元素的数据类型是 DT_UINT32, 否则是 DT_INT32。
- shift 的元素的数据类型是 DT_INT8, 取值范围是 [-32, 0], 移位只有当 left_dtype 和 right_dtype 是 DT_INT32 或 DT_UINT32 时生效。

4.15.5 tpu_bdc_int_pcs_mm_L_trans

左矩阵的转置乘以右矩阵, 对结果按 column 做算术移位。

```
void tpu_bdc_int_pcs_mm_L_trans(local_addr_t output_addr, local_addr_t
                                left_addr, local_addr_t right_addr, local_addr_t
                                shift_addr, int left_rows, int left_cols, int
                                right_cols, int left_cols_per_channel, int
                                right_cols_per_channel, data_type_t left_dtype,
                                data_type_t right_dtype, rounding_mode_t
                                rounding_mode)
```

$$\text{output}_{M \times N} = (\text{left}_{K \times M}^T \times \text{right}_{K \times N}) \text{ 右移 } - \text{shift}_{1 \times N}$$

$$\text{output}(m, n) = \left(\sum_k \text{left}(k, m) \times \text{right}(k, n) \right) \text{ 右移 } - \text{shift}(0, n)$$

参数

- output_addr - output 的地址
- left_addr - left 的地址
- right_addr - right 的地址

- `shift_addr` - shift 的地址
- `left_rows` - 左矩阵的行数
- `left_cols` - 左矩阵的列数
- `right_cols` - 右矩阵的列数
- `left_cols_per_channel` - 左矩阵在每个 channel 的列数
- `right_cols_per_channel` - 右矩阵在每个 channel 的列数
- `left_dtype` - left 的元素的数据类型
- `right_dtype` - right 的元素的数据类型
- `rounding_mode` - 右移舍入模式

注意事项

- `output`、`right` 和 `shift` 从同一个 NPU 开始。
- `left_rows`、`left_cols` 和 `right_cols` 的取值范围是 [1, 65535]。
- `output`、`left`、`right` 和 `shift` 是 matrix layout, `output` 的行数是 `left_cols`, 列数是 `right_cols`, `right` 的行数是 `left_rows`, `shift` 的行数是 1, 列数是 `right_cols`。
- `left_dtype` 和 `right_dtype` 的位宽相同, 有效取值是 `DT_INT32`、`DT_UINT32`、`DT_INT16`、`DT_UINT16`、`DT_INT8` 和 `DT_UINT8`, 如果 `left_dtype` 和 `right_dtype` 都是无符号的, 则 `output` 的元素的数据类型是 `DT_UINT32`, 否则是 `DT_INT32`。
- `shift` 的元素的数据类型是 `DT_INT8`, 取值范围是 [-32, 0], 移位只有当 `left_dtype` 和 `right_dtype` 是 `DT_INT32` 或 `DT_UINT32` 时生效。

4.15.6 tpu_bdc_int_pcs_mm_L_const

两个矩阵相乘, 其中左矩阵的所有元素都相同, 对结果按 column 做算术移位。

```
void tpu_bdc_int_pcs_mm_L_const(local_addr_t output_addr, local_addr_t
                                right_addr, local_addr_t shift_addr, scalar_t C,
                                int left_rows, int left_cols, int right_cols, int
                                right_cols_per_channel, data_type_t C_dtype,
                                data_type_t right_dtype, rounding_mode_t
                                rounding_mode)
```

$$\text{output}_{M \times N} = (\text{left}_{M \times K} \times \text{right}_{K \times N}) \text{ 右移 } - \text{shift}_{1 \times N}$$

$$\text{output}(m, n) = \left(\sum_k C \times \text{right}(k, n) \right) \text{ 右移 } - \text{shift}(0, n)$$

参数

- `output_addr` - output 的地址
- `right_addr` - right 的地址
- `shift_addr` - shift 的地址
- `C` - 常数
- `left_rows` - 左矩阵的行数
- `left_cols` - 左矩阵的列数
- `right_cols` - 右矩阵的列数
- `right_cols_per_channel` - 右矩阵在每个 channel 的列数
- `C_dtype` - C 的数据类型

- `right_dtype` -right 的元素的数据类型
- `rounding_mode` -右移舍入模式

注意事项

- `output`、`right` 和 `shift` 从同一个 NPU 开始。
- `left_rows`、`left_cols` 和 `right_cols` 的取值范围是 [1, 65535]。
- `output`、`right` 和 `shift` 是 matrix layout, `output` 的行数是 `left_rows`, 列数是 `right_cols`, `right` 的行数是 `left_cols`, `shift` 的行数是 1, 列数是 `right_cols`。
- `C_dtype` 和 `right_dtype` 的位宽相同, 有效取值是 `DT_INT32`、`DT_UINT32`、`DT_INT16`、`DT_UINT16`、`DT_INT8` 和 `DT_UINT8`, 如果 `C_dtype` 和 `right_dtype` 都是无符号的, 则 `output` 的元素的数据类型是 `DT_UINT32`, 否则是 `DT_INT32`。
- `shift` 的元素的数据类型是 `DT_INT8`, 取值范围是 [-32, 0], 移位只有当 `C_dtype` 和 `right_dtype` 是 `DT_INT32` 或 `DT_UINT32` 时生效。

4.15.7 tpu_bdc_int8_mm

两个矩阵相乘, 对结果做累加 (可选, 在累加之前会对结果的原数据算数左移), 再对结果按 column 加 bias (可选), 再对结果做 ReLU (可选), 最后对结果做算数右移, 结果有 saturation。

```
void tpu_bdc_int8_mm(local_addr_t output_addr, local_addr_t left_addr,
                    local_addr_t right_addr, local_addr_t bias_addr, int
                    left_rows, int left_cols, int right_cols, int
                    left_cols_per_channel, int right_cols_per_channel,
                    data_type_t output_dtype, data_type_t left_dtype,
                    data_type_t right_dtype, data_type_t bias_dtype, unsigned
                    char lshift, unsigned char rshift, bool has_bias, bool result_add,
                    bool result_relu)
```

$$\text{output}_{M \times N} = \text{ReLU}((\text{output}_{M \times N} \text{ 左移 } lshift) + \text{left}_{M \times K} \times \text{right}_{K \times N} + \text{bias}_{1 \times N}) \text{ 右移 } rshift$$

$$\text{output}(m, n) = \text{ReLU}(((\text{output}(m, n) \text{ 左移 } lshift) + (\sum_k \text{left}(m, k) \times \text{right}(k, n)) + \text{bias}(0, n))) \text{ 右移 } rshift$$

参数

- `output_addr` -output 的地址
- `left_addr` -left 的地址
- `right_addr` -right 的地址
- `bias_addr` -bias 的地址
- `left_rows` -左矩阵的行数
- `left_cols` -左矩阵的列数
- `right_cols` -右矩阵的列数
- `left_cols_per_channel` -左矩阵在每个 channel 的列数
- `right_cols_per_channel` -右矩阵在每个 channel 的列数
- `output_dtype` -output 的元素的数据类型
- `left_dtype` -left 的元素的数据类型
- `right_dtype` -right 的元素的数据类型
- `bias_dtype` -bias 的元素的数据类型
- `lshift` -左移位数

- `rshift` - 右移位数
- `has_bias` - 加 `bias` 的标志
- `result_add` - 对结果做累加的标志
- `result_relu` - 对结果做 ReLU 的标志

注意事项

- `output`、`right` 和 `bias` 从同一个 NPU 开始。
- `left_rows`、`left_cols` 和 `right_cols` 的取值范围是 `[1, 65535]`。
- `output`、`left`、`right` 和 `bias` 是 matrix layout, `output` 的行数是 `left_rows`, 列数是 `right_cols`, `right` 的行数是 `left_cols`, `bias` 的行数是 1, 列数是 `right_cols`。
- `output_dtype` 的有效取值是 `DT_INT16`、`DT_UINT16`、`DT_INT8` 和 `DT_UINT8`, `left_dtype` 和 `right_dtype` 的有效取值是 `DT_INT8` 和 `DT_UINT8`, `bias_dtype` 的有效取值是 `DT_INT16` 和 `DT_UINT16`。
- 如果 `left_dtype`、`right_dtype` 和 `bias_dtype` (如果加 `bias` 的话) 都是无符号的, 则 `output` 累加前的元素的数据类型被认定为无符号的, 否则是有符号的。
- 如果 `left_dtype`、`right_dtype` 和 `bias_dtype` (如果加 `bias` 的话) 都是无符号的, 或 `result_relu` 是 `true`, 则 `output` 的元素的数据类型被认定为无符号的, 否则是有符号的。
- 如果 `output_dtype` 是 `DT_INT8` 或 `DT_UINT8`, 对结果做累加, 则累加前的 `output` 的元素的数据类型会被认定为 `DT_INT16` 或 `DT_UINT16`, 而不是 `DT_INT8` 或 `DT_UINT8`。
- `lshift` 和 `rshift` 的取值范围是 `[0, 31]`。

4.15.8 tpu_bdc_int8_mm_L_trans

左矩阵的转置乘以右矩阵, 对结果做累加 (可选, 在累加之前会对结果的原数据算数左移), 再对结果按 column 加 `bias` (可选), 再对结果做 ReLU (可选), 最后对结果做算数右移, 结果有 `saturation`。

```
void tpu_bdc_int8_mm_L_trans(local_addr_t output_addr, local_addr_t left_addr,
                             local_addr_t right_addr, local_addr_t bias_addr, int
                             left_rows, int left_cols, int right_cols, int
                             left_cols_per_channel, int right_cols_per_channel,
                             data_type_t output_dtype, data_type_t left_dtype,
                             data_type_t right_dtype, data_type_t bias_dtype,
                             unsigned char lshift, unsigned char rshift, bool
                             has_bias, bool result_add, bool result_relu)
```

$$\text{output}_{M \times N} = \text{ReLU}((\text{output}_{M \times N} \text{ 左移 } lshift) + \text{left}_{K \times M}^T \times \text{right}_{K \times N} + \text{bias}_{1 \times N}) \text{ 右移 } rshift$$

$$\text{output}(m, n) = \text{ReLU}(((\text{output}(m, n) \text{ 左移 } lshift) + (\sum_k \text{left}(k, m) \times \text{right}(k, n)) + \text{bias}(0, n))) \text{ 右移 } rshift$$

参数

- `output_addr` - `output` 的地址
- `left_addr` - `left` 的地址
- `right_addr` - `right` 的地址
- `bias_addr` - `bias` 的地址
- `left_rows` - 左矩阵的行数
- `left_cols` - 左矩阵的列数
- `right_cols` - 右矩阵的列数

- `left_cols_per_channel` - 左矩阵在每个 channel 的列数
- `right_cols_per_channel` - 右矩阵在每个 channel 的列数
- `output_dtype` - output 的元素的数据类型
- `left_dtype` - left 的元素的数据类型
- `right_dtype` - right 的元素的数据类型
- `bias_dtype` - bias 的元素的数据类型
- `lshift` - 左移位数
- `rshift` - 右移位数
- `has_bias` - 加 bias 的标志
- `result_add` - 对结果做累加的标志
- `result_relu` - 对结果做 ReLU 的标志

注意事项

- output、right 和 bias 从同一个 NPU 开始。
- `left_rows`、`left_cols` 和 `right_cols` 的取值范围是 [1, 65535]。
- output、left、right 和 bias 是 matrix layout, output 的行数是 `left_cols`, 列数是 `right_cols`, right 的行数是 `left_rows`, bias 的行数是 1, 列数是 `right_cols`。
- `output_dtype` 的有效取值是 `DT_INT16`、`DT_UINT16`、`DT_INT8` 和 `DT_UINT8`, `left_dtype` 和 `right_dtype` 的有效取值是 `DT_INT8` 和 `DT_UINT8`, `bias_dtype` 的有效取值是 `DT_INT16` 和 `DT_UINT16`。
- 如果 `left_dtype`、`right_dtype` 和 `bias_dtype` (如果加 bias 的话) 都是无符号的, 则 output 累加前的元素的数据类型被认定为无符号的, 否则是有符号的。
- 如果 `left_dtype`、`right_dtype` 和 `bias_dtype` (如果加 bias 的话) 都是无符号的, 或 `result_relu` 是 true, 则 output 的元素的数据类型被认定为无符号的, 否则是有符号的。
- 如果 `output_dtype` 是 `DT_INT8` 或 `DT_UINT8`, 对结果做累加, 则累加前的 output 的元素的数据类型会被认定为 `DT_INT16` 或 `DT_UINT16`, 而不是 `DT_INT8` 或 `DT_UINT8`。
- `lshift` 和 `rshift` 的取值范围是 [0, 31]。

4.15.9 tpu_bdc_int8_mm_L_const

两个矩阵相乘, 对结果做累加 (可选, 在累加之前会对结果的原数据算数左移), 再对结果按 column 加 bias (可选), 再对结果做 ReLU (可选), 最后对结果做算数右移, 结果有 saturation。

```
void tpu_bdc_int8_mm_L_const(local_addr_t output_addr, local_addr_t right_addr,
                             local_addr_t bias_addr, scalar_t C, int left_rows, int
                             left_cols, int right_cols, int right_cols_per_channel,
                             data_type_t output_dtype, data_type_t C_dtype,
                             data_type_t right_dtype, data_type_t bias_dtype,
                             unsigned char lshift, unsigned char rshift, bool
                             has_bias, bool result_add, bool result_relu)
```

$$\text{output}_{M \times N} = \text{ReLU}((\text{output}_{M \times N} \text{ 左移 } lshift) + \text{left}_{M \times K} \times \text{right}_{K \times N} + \text{bias}_{1 \times N}) \text{ 右移 } rshift$$

$$\text{output}(m, n) = \text{ReLU}(((\text{output}(m, n) \text{ 左移 } lshift) + (\sum_k C \times \text{right}(k, n)) + \text{bias}(0, n))) \text{ 右移 } rshift$$

参数

- `output_addr` - output 的地址
- `right_addr` - right 的地址
- `bias_addr` - bias 的地址
- `C` - 常数
- `left_rows` - 左矩阵的行数
- `left_cols` - 左矩阵的列数
- `right_cols` - 右矩阵的列数
- `right_cols_per_channel` - 右矩阵在每个 channel 的列数
- `output_dtype` - output 的元素的数据类型
- `C_dtype` - C 的数据类型
- `right_dtype` - right 的元素的数据类型
- `bias_dtype` - bias 的元素的数据类型
- `lshift` - 左移位数
- `rshift` - 右移位数
- `has_bias` - 加 bias 的标志
- `result_add` - 对结果做累加的标志
- `result_relu` - 对结果做 ReLU 的标志

注意事项

- output、right 和 bias 从同一个 NPU 开始。
- `left_rows`、`left_cols` 和 `right_cols` 的取值范围是 [1, 65535]。
- output、right 和 bias 是 matrix layout, output 的行数是 `left_rows`, 列数是 `right_cols`, right 的行数是 `left_cols`, bias 的行数是 1, 列数是 `right_cols`。
- `output_dtype` 的有效取值是 `DT_INT16`、`DT_UINT16`、`DT_INT8` 和 `DT_UINT8`, `C_dtype` 和 `right_dtype` 的有效取值是 `DT_INT8` 和 `DT_UINT8`, `bias_dtype` 的有效取值是 `DT_INT16` 和 `DT_UINT16`。
- 如果 `C_dtype`、`right_dtype` 和 `bias_dtype` (如果加 bias 的话) 都是无符号的, 则 output 累加前的元素的数据类型被认定为无符号的, 否则是有符号的。
- 如果 `C_dtype`、`right_dtype` 和 `bias_dtype` (如果加 bias 的话) 都是无符号的, 或 `result_relu` 是 true, 则 output 的元素的数据类型被认定为无符号的, 否则是有符号的。
- 如果 `output_dtype` 是 `DT_INT8` 或 `DT_UINT8`, 对结果做累加, 则累加前的 output 的元素的数据类型会被认定为 `DT_INT16` 或 `DT_UINT16`, 而不是 `DT_INT8` 或 `DT_UINT8`。
- `lshift` 和 `rshift` 的取值范围是 [0, 31]。

4.15.10 tpu_bdc_int8_zp_mm

两个矩阵相乘, 其中右矩阵的元素减 zero-point, 再对结果做累加 (可选), 结果没有 saturation。

```
void tpu_bdc_int8_zp_mm(local_addr_t output_addr, local_addr_t left_addr,
                        local_addr_t right_addr, scalar_t zp_val, int left_rows, int
                        left_cols, int right_cols, data_type_t left_dtype,
                        data_type_t right_zp_dtype, bool result_add)
```

$$\text{output}_{M \times N} = \text{output}_{M \times N} + \text{left}_{M \times K} \times (\text{right}_{K \times N} - \text{zp})$$

$$\text{output}(m, n) = \text{output}(m, n) + \sum_k \text{left}(m, k) \times (\text{right}(k, n) - \text{zp})$$

参数

- `output_addr` -output 的地址
- `left_addr` -left 的地址
- `right_addr` -right 的地址
- `zp_val` -zero-point 的值
- `left_rows` -左矩阵的行数
- `left_cols` -左矩阵的列数
- `right_cols` -右矩阵的列数
- `left_dtype` -left 的元素的数据类型
- `right_zp_dtype` -right 的元素和 `zp_val` 的数据类型
- `result_add` -对结果做累加的标志

注意事项

- `output`、`left` 和 `right` 从 NPU 0 开始，都是 64-byte aligned layout。
- `output` 的 shape 是 `[1, left_rows, 1, right_cols]`，`left` 的 shape 是 `[1, left_rows, 1, left_cols]`，`right` 的 shape 是 `[1, left_cols, 1, right_cols]`。
- `left_rows`、`left_cols` 和 `right_cols` 的取值范围是 `[1, 65535]`。
- `output` 的元素的数据类型是 `DT_INT32`，`left_dtype` 和 `right_zp_dtype` 的有效取值是 `DT_INT8` 和 `DT_UINT8`。

4.15.11 tpu_bdc_int8_zp_mm_R_trans

左矩阵乘以右矩阵的转置，其中右矩阵的元素减 zero-point，结果没有 saturation。

```
void tpu_bdc_int8_zp_mm_R_trans(local_addr_t output_addr, local_addr_t
                                left_addr, local_addr_t right_addr, scalar_t
                                zp_val, int left_rows, int left_cols, int
                                right_rows, data_type_t left_dtype,
                                data_type_t right_zp_dtype)
```

$$\text{output}_{M \times N} = \text{left}_{M \times K} \times (\text{right}_{N \times K}^T - \text{zp})$$

$$\text{output}(m, n) = \sum_k \text{left}(m, k) \times (\text{right}(n, k) - \text{zp})$$

参数

- `output_addr` -output 的地址
- `left_addr` -left 的地址
- `right_addr` -right 的地址
- `zp_val` -zero-point 的值
- `left_rows` -左矩阵的行数
- `left_cols` -左矩阵的列数
- `right_rows` -右矩阵的行数
- `left_dtype` -left 的元素的数据类型

- `right_zp_dtype` -right 的元素和 `zp_val` 的数据类型

注意事项

- `output`、`left` 和 `right` 从 NPU 0 开始，都是 64-byte aligned layout。
- `output` 的 shape 是 `[1, left_rows, 1, right_rows]`，`left` 的 shape 是 `[1, left_rows, 1, left_cols]`，`right` 的 shape 是 `[1, right_rows, 1, left_cols]`。
- `left_rows`、`left_cols` 和 `right_rows` 的取值范围是 `[1, 65535]`。
- `output` 的元素的数据类型是 `DT_INT32`，`left_dtype` 和 `right_zp_dtype` 的有效取值是 `DT_INT8` 和 `DT_UINT8`。

4.15.12 tpu_bdc_int8_zp_mm_all_trans

两个矩阵的转置相乘，结果也转置，其中右矩阵的元素减 zero-point，再对结果做累加（可选），结果没有 saturation。

```
void tpu_bdc_int8_zp_mm_all_trans(local_addr_t output_addr, local_addr_t
                                left_addr, local_addr_t right_addr, scalar_t
                                zp_val, int left_rows, int left_cols, int
                                right_rows, data_type_t left_dtype,
                                data_type_t right_zp_dtype, bool result_add)
```

$$\text{output}_{N \times M}^T = \text{output}_{N \times M}^T + \text{left}_{K \times M}^T \times (\text{right}_{N \times K}^T - \text{zp})$$

$$\text{output}(n, m) = \text{output}(n, m) + \sum_k \text{left}(k, m) \times (\text{right}(n, k) - \text{zp})$$

参数

- `output_addr` -output 的地址
- `left_addr` -left 的地址
- `right_addr` -right 的地址
- `zp_val` -zero-point 的值
- `left_rows` -左矩阵的行数
- `left_cols` -左矩阵的列数
- `right_rows` -右矩阵的行数
- `left_dtype` -left 的元素的数据类型
- `right_zp_dtype` -right 的元素和 `zp_val` 的数据类型
- `result_add` -对结果做累加的标志

注意事项

- `output`、`left` 和 `right` 从 NPU 0 开始，都是 64-byte aligned layout。
- `output` 的 shape 是 `[1, right_rows, 1, left_cols]`，`left` 的 shape 是 `[1, left_rows, 1, left_cols]`，`right` 的 shape 是 `[1, right_rows, 1, left_rows]`。
- `left_rows`、`left_cols` 和 `right_rows` 的取值范围是 `[1, 65535]`。
- `output` 的元素的数据类型是 `DT_INT32`，`left_dtype` 和 `right_zp_dtype` 的有效取值是 `DT_INT8` 和 `DT_UINT8`。

4.15.13 tpu_bdc_int8_zp_mm_L_const

两个矩阵相乘，其中左矩阵的所有元素都相同，右矩阵的元素减 zero-point，再对结果做累加（可选），结果没有 saturation。

```
void tpu_bdc_int8_zp_mm_L_const(local_addr_t output_addr, local_addr_t
                                right_addr, scalar_t C, scalar_t zp_val, int
                                left_rows, int left_cols, int right_cols,
                                data_type_t C_dtype, data_type_t
                                right_zp_dtype, bool result_add)
```

$$\text{output}_{M \times N} = \text{output}_{M \times N} + \text{left}_{M \times K} \times (\text{right}_{K \times N} - \text{zp})$$

$$\text{output}(m, n) = \text{output}(m, n) + \sum_k C \times (\text{right}(k, n) - \text{zp})$$

参数

- output_addr - output 的地址
- right_addr - right 的地址
- C - 左矩阵元素值
- zp_val - zero-point 的值
- left_rows - 左矩阵的行数
- left_cols - 左矩阵的列数
- right_cols - 右矩阵的列数
- C_dtype - C 的数据类型
- right_zp_dtype - right 的元素和 zp_val 的数据类型
- result_add - 对结果做累加的标志

注意事项

- output 和 right 从 NPU 0 开始，都是 64-byte aligned layout。
- output 的 shape 是 [1, left_rows, 1, right_cols]，left 的 shape 是 [1, left_rows, 1, left_cols]，right 的 shape 是 [1, left_cols, 1, right_cols]。
- left_rows、left_cols 和 right_cols 的取值范围是 [1, 65535]。
- output 的元素的数据类型是 DT_INT32，C_dtype 和 right_zp_dtype 的有效取值是 DT_INT8 和 DT_UINT8。

4.15.14 tpu_bdc_int8_zp_mm_R_const

两个矩阵相乘，其中右矩阵的所有元素都相同，右矩阵的元素减 zero-point，再对结果做累加（可选），结果没有 saturation。

```
void tpu_bdc_int8_zp_mm_R_const(local_addr_t output_addr, local_addr_t
                                left_addr, scalar_t C, scalar_t zp_val, int
                                left_rows, int left_cols, int right_cols,
                                data_type_t left_dtype, data_type_t
                                C_zp_dtype, bool result_add)
```

$$\text{output}_{M \times N} = \text{output}_{M \times N} + \text{left}_{M \times K} \times (\text{right}_{K \times N} - \text{zp})$$

$$\text{output}(m, n) = \text{output}(m, n) + \sum_k \text{left}(m, k) \times (C - \text{zp})$$

参数

- output_addr - output 的地址
- left_addr - left 的地址
- C - 右矩阵元素值
- zp_val - zero-point 的值
- left_rows - 左矩阵的行数
- left_cols - 左矩阵的列数
- right_cols - 右矩阵的列数
- left_dtype - left 的元素的数据类型
- C_zp_dtype - C 和 zp_val 的数据类型
- result_add - 对结果做累加的标志

注意事项

- output 和 left 从 NPU 0 开始，都是 64-byte aligned layout。
- output 的 shape 是 [1, left_rows, 1, right_cols], left 的 shape 是 [1, left_rows, 1, left_cols], right 的 shape 是 [1, left_cols, 1, right_cols]。
- left_rows、left_cols 和 right_cols 的取值范围是 [1, 65535]。
- output 的元素的数据类型是 DT_INT32, left_dtype 和 C_zp_dtype 的有效取值是 DT_INT8 和 DT_UINT8。

4.15.15 tpu_bdc_int8_zp_mm_L_const_R_trans

左矩阵乘以右矩阵的转置，其中左矩阵的所有元素都相同，右矩阵的元素减 zero-point，结果没有 saturation。

```
void tpu_bdc_int8_zp_mm_L_const_R_trans(local_addr_t output_addr,
                                         local_addr_t right_addr, scalar_t C,
                                         scalar_t zp_val, int left_rows, int
                                         left_cols, int right_rows, data_type_t
                                         C_dtype, data_type_t
                                         right_zp_dtype)
```

$$\text{output}_{M \times N} = \text{left}_{M \times K} \times (\text{right}_{N \times K}^T - \text{zp})$$

$$\text{output}(m, n) = \sum_k C \times (\text{right}(n, k) - \text{zp})$$

参数

- output_addr - output 的地址
- right_addr - right 的地址
- C - 左矩阵元素值
- zp_val - zero-point 的值
- left_rows - 左矩阵的行数
- left_cols - 左矩阵的列数
- right_rows - 右矩阵的行数
- C_dtype - C 的数据类型

- `right_zp_dtype` -right 的元素和 `zp_val` 的数据类型

注意事项

- output 和 right 从 NPU 0 开始，都是 64-byte aligned layout。
- output 的 shape 是 [1, left_rows, 1, right_rows], left 的 shape 是 [1, left_rows, 1, left_cols], right 的 shape 是 [1, right_rows, 1, left_cols]。
- left_rows、left_cols 和 right_rows 的取值范围是 [1, 65535]。
- output 的元素的数据类型是 `DT_INT32`, `C_dtype` 和 `right_zp_dtype` 的有效取值是 `DT_INT8` 和 `DT_UINT8`。

4.15.16 tpu_bdc_int8_zp_mm_L_const_all_trans

两个矩阵的转置相乘，其中左矩阵的所有元素都相同，右矩阵的元素减 zero-point，结果也转置，再对结果做累加（可选），结果没有 saturation。

```
void tpu_bdc_int8_zp_mm_L_const_all_trans(local_addr_t output_addr,
                                           local_addr_t right_addr, scalar_t C,
                                           scalar_t zp_val, int left_rows, int
                                           left_cols, int right_rows, data_type_t
                                           C_dtype, data_type_t
                                           right_zp_dtype, bool result_add)
```

$$\text{output}_{N \times M}^T = \text{output}_{N \times M}^T + \text{left}_{K \times M}^T \times (\text{right}_{N \times K}^T - \text{zp})$$

$$\text{output}(n, m) = \text{output}(n, m) + \sum_k C \times (\text{right}(n, k) - \text{zp})$$

参数

- `output_addr` -output 的地址
- `right_addr` -right 的地址
- `C` -左矩阵元素值
- `zp_val` -zero-point 的值
- `left_rows` -左矩阵的行数
- `left_cols` -左矩阵的列数
- `right_rows` -右矩阵的行数
- `C_dtype` -C 的数据类型
- `right_zp_dtype` -right 的元素和 `zp_val` 的数据类型
- `result_add` -对结果做累加的标志

注意事项

- output 和 right 从 NPU 0 开始，都是 64-byte aligned layout。
- output 的 shape 是 [1, right_rows, 1, left_cols], left 的 shape 是 [1, left_rows, 1, left_cols], right 的 shape 是 [1, right_rows, 1, left_rows]。
- left_rows、left_cols 和 right_rows 的取值范围是 [1, 65535]。
- output 的元素的数据类型是 `DT_INT32`, `C_dtype` 和 `right_zp_dtype` 的有效取值是 `DT_INT8` 和 `DT_UINT8`。

4.15.17 tpu_bdc_int8_zp_mm_R_const_all_trans

两个矩阵的转置相乘，其中右矩阵的所有元素都相同，右矩阵的元素减 zero-point，结果也转置，再对结果做累加（可选），结果没有 saturation。

```
void tpu_bdc_int8_zp_mm_R_const_all_trans(local_addr_t output_addr,
                                           local_addr_t left_addr, scalar_t C,
                                           scalar_t zp_val, int left_rows, int
                                           left_cols, int right_rows, data_type_t
                                           left_dtype, data_type_t C_zp_dtype,
                                           bool result_add)
```

$$\text{output}_{N \times M}^T = \text{output}_{N \times M}^T + \text{left}_{K \times M}^T \times (\text{right}_{N \times K}^T - \text{zp})$$

$$\text{output}(n, m) = \text{output}(n, m) + \sum_k \text{left}(k, m) \times (C - \text{zp})$$

参数

- output_addr - output 的地址
- left_addr - left 的地址
- C - 右矩阵元素值
- zp_val - zero-point 的值
- left_rows - 左矩阵的行数
- left_cols - 左矩阵的列数
- right_rows - 右矩阵的行数
- left_dtype - left 的元素的数据类型
- C_zp_dtype - C 和 zp_val 的数据类型
- result_add - 对结果做累加的标志

注意事项

- output 和 left 从 NPU 0 开始，都是 64-byte aligned layout。
- output 的 shape 是 [1, right_rows, 1, left_cols]，left 的 shape 是 [1, left_rows, 1, left_cols]，right 的 shape 是 [1, right_rows, 1, left_rows]。
- left_rows、left_cols 和 right_rows 的取值范围是 [1, 65535]。
- output 的元素的数据类型是 DT_INT32，left_dtype 和 C_zp_dtype 的有效取值是 DT_INT8 和 DT_UINT8。

4.15.18 tpu_bdc_int8_pc_zp_mm

两个矩阵相乘，其中右矩阵的元素按 column 减 zero-point，再对结果做累加（可选），结果没有 saturation。

```
void tpu_bdc_int8_pc_zp_mm(local_addr_t output_addr, local_addr_t left_addr,
                           local_addr_t right_addr, local_addr_t zp_addr, int
                           left_rows, int left_cols, int right_cols, data_type_t
                           left_dtype, data_type_t right_zp_dtype, bool
                           result_add)
```

$$\text{output}_{M \times N} = \text{output}_{M \times N} + \text{left}_{M \times K} \times (\text{right}_{K \times N} - \text{zp}_{1 \times N})$$

$$\text{output}(m, n) = \text{output}(m, n) + \sum_k \text{left}(m, k) \times (\text{right}(k, n) - \text{zp}(0, n))$$

参数

- output_addr -output 的地址
- left_addr -left 的地址
- right_addr -right 的地址
- zp_addr -zp 的地址
- left_rows -左矩阵的行数
- left_cols -左矩阵的列数
- right_cols -右矩阵的列数
- left_dtype -left 的元素的数据类型
- right_zp_dtype -right 和 zp 的元素的数据类型
- result_add -对结果做累加的标志

注意事项

- output、left、right 和 zp 从 NPU 0 开始，都是 64-byte aligned layout。
- output 的 shape 是 [1, left_rows, 1, right_cols], left 的 shape 是 [1, left_rows, 1, left_cols], right 的 shape 是 [1, left_cols, 1, right_cols], zp 的 shape 是 [1, NPU_NUM, 1, right_cols], 而不是 [1, 1, 1, right_cols], 每个 NPU 中都有一份相同的。
- left_rows、left_cols 和 right_cols 的取值范围是 [1, 65535]。
- output 的元素的数据类型是DT_INT32, left_dtype 和 right_zp_dtype 的有效取值是DT_INT8 和DT_UINT8。

4.15.19 tpu_bdc_int8_pc_zp_mm_R_trans

左矩阵乘以右矩阵的转置,其中转置后的右矩阵的元素按 column 减 zero-point,结果没有 saturation。

```
void tpu_bdc_int8_pc_zp_mm_R_trans(local_addr_t output_addr, local_addr_t
                                   left_addr, local_addr_t right_addr,
                                   local_addr_t zp_addr, int left_rows, int
                                   left_cols, int right_rows, data_type_t
                                   left_dtype, data_type_t right_zp_dtype)
```

$$\text{output}_{M \times N} = \text{left}_{M \times K} \times (\text{right}_{N \times K}^T - \text{zp}_{1 \times N})$$

$$\text{output}(m, n) = \sum_k \text{left}(m, k) \times (\text{right}(n, k) - \text{zp}(0, n))$$

参数

- output_addr -output 的地址
- left_addr -left 的地址
- right_addr -right 的地址
- zp_addr -zp 的地址
- left_rows -左矩阵的行数
- left_cols -左矩阵的列数
- right_rows -右矩阵的行数
- left_dtype -left 的元素的数据类型
- right_zp_dtype -right 和 zp 的元素的数据类型

注意事项

- output、left、right 和 zp 从 NPU 0 开始，都是 64-byte aligned layout。
- output 的 shape 是 [1, left_rows, 1, right_rows], left 的 shape 是 [1, left_rows, 1, left_cols], right 的 shape 是 [1, right_rows, 1, left_cols], zp 的 shape 是 [1, NPU_NUM, 1, right_rows], 而不是 [1, 1, 1, right_rows], 每个 NPU 中都有一份相同的。
- left_rows、left_cols 和 right_rows 的取值范围是 [1, 65535]。
- output 的元素的数据类型是DT_INT32, left_dtype 和 right_zp_dtype 的有效取值是DT_INT8 和DT_UINT8。

4.15.20 tpu_bdc_int8_pc_zp_mm_all_trans

两个矩阵的转置相乘，结果也转置，其中转置前的右矩阵的元素按 row 减 zero-point，再对结果做累加（可选），结果没有 saturation。

```
void tpu_bdc_int8_pc_zp_mm_all_trans(local_addr_t output_addr, local_addr_t
                                     left_addr, local_addr_t right_addr,
                                     local_addr_t zp_addr, int left_rows, int
                                     left_cols, int right_rows, data_type_t
                                     left_dtype, data_type_t right_zp_dtype,
                                     bool result_add)
```

$$\text{output}_{N \times M}^T = \text{output}_{N \times M}^T + \text{left}_{K \times M}^T \times (\text{right}_{N \times K} - \text{zp}_{N \times 1})^T$$

$$\text{output}(n, m) = \text{output}(n, m) + \sum_k \text{left}(k, m) \times (\text{right}(n, k) - \text{zp}(n, 0))$$

参数

- output_addr - output 的地址
- left_addr - left 的地址
- right_addr - right 的地址
- zp_addr - zp 的地址
- left_rows - 左矩阵的行数
- left_cols - 左矩阵的列数
- right_rows - 右矩阵的行数
- left_dtype - left 的元素的数据类型
- right_zp_dtype - right 的元素和 zp_val 的数据类型
- result_add - 对结果做累加的标志

注意事项

- output、left、right 和 zp 从 NPU 0 开始，output、left 和 right 是 64-byte aligned layout, zp 是 compact layout。
- output 的 shape 是 [1, right_rows, 1, left_cols], left 的 shape 是 [1, left_rows, 1, left_cols], right 的 shape 是 [1, right_rows, 1, left_rows], zp 的 shape 是 [1, right_rows, 1, 1]。
- left_rows、left_cols 和 right_rows 的取值范围是 [1, 65535]。
- output 的元素的数据类型是DT_INT32, left_dtype 和 right_zp_dtype 的有效取值是DT_INT8 和DT_UINT8。

4.15.21 tpu_bdc_int8_pc_zp_mm_L_const

两个矩阵相乘，其中左矩阵的所有元素都相同，右矩阵的元素按 column 减 zero-point，再对结果做累加（可选），结果没有 saturation。

```
void tpu_bdc_int8_pc_zp_mm_L_const(local_addr_t output_addr, local_addr_t
                                   right_addr, local_addr_t zp_addr, scalar_t
                                   C, int left_rows, int left_cols, int right_cols,
                                   data_type_t C_dtype, data_type_t
                                   right_zp_dtype, bool result_add)
```

$$\text{output}_{M \times N} = \text{output}_{M \times N} + \text{left}_{M \times K} \times (\text{right}_{K \times N} - \text{zp}_{1 \times N})$$

$$\text{output}(m, n) = \text{output}(m, n) + \sum_k C \times (\text{right}(k, n) - \text{zp}(0, n))$$

参数

- output_addr - output 的地址
- right_addr - right 的地址
- zp_addr - zp 的地址
- C - 左矩阵元素值
- left_rows - 左矩阵的行数
- left_cols - 左矩阵的列数
- right_cols - 右矩阵的列数
- C_dtype - C 的数据类型
- right_zp_dtype - right 的元素和 zp_val 的数据类型
- result_add - 对结果做累加的标志

注意事项

- output、right 和 zp 从 NPU 0 开始，都是 64-byte aligned layout。
- output 的 shape 是 [1, left_rows, 1, right_cols]，left 的 shape 是 [1, left_rows, 1, left_cols]，right 的 shape 是 [1, left_cols, 1, right_cols]，zp 的 shape 是 [1, NPU_NUM, 1, right_cols]，而不是 [1, 1, 1, right_cols]，每个 NPU 中都有一份相同的。
- left_rows、left_cols 和 right_cols 的取值范围是 [1, 65535]。
- output 的元素的数据类型是 DT_INT32，C_dtype 和 right_zp_dtype 的有效取值是 DT_INT8 和 DT_UINT8。

4.15.22 tpu_bdc_int8_pc_zp_mm_R_const

两个矩阵相乘，其中右矩阵的所有元素都相同，右矩阵的元素按 column 减 zero-point，再对结果做累加（可选），结果没有 saturation。

```
void tpu_bdc_int8_pc_zp_mm_R_const(local_addr_t output_addr, local_addr_t
                                   left_addr, local_addr_t zp_addr, scalar_t C,
                                   int left_rows, int left_cols, int right_cols,
                                   data_type_t left_dtype, data_type_t
                                   C_zp_dtype, bool result_add)
```

$$\text{output}_{M \times N} = \text{output}_{M \times N} + \text{left}_{M \times K} \times (\text{right}_{K \times N} - \text{zp}_{1 \times N})$$

$$\text{output}(m, n) = \text{output}(m, n) + \sum_k \text{left}(m, k) \times (C - \text{zp}(0, n))$$

参数

- output_addr - output 的地址
- left_addr - left 的地址
- zp_addr - zp 的地址
- C - 右矩阵元素值
- left_rows - 左矩阵的行数
- left_cols - 左矩阵的列数
- right_cols - 右矩阵的列数
- left_dtype - left 的元素的数据类型
- C_zp_dtype - zp 的元素和 C 的数据类型
- result_add - 对结果做累加的标志

注意事项

- output、left 和 zp 从 NPU 0 开始，都是 64-byte aligned layout。
- output 的 shape 是 [1, left_rows, 1, right_cols], left 的 shape 是 [1, left_rows, 1, left_cols], right 的 shape 是 [1, left_cols, 1, right_cols], zp 的 shape 是 [1, NPU_NUM, 1, right_cols], 而不是 [1, 1, 1, right_cols], 每个 NPU 中都有一份相同的。
- left_rows、left_cols 和 right_cols 的取值范围是 [1, 65535]。
- output 的元素的数据类型是 DT_INT32, left_dtype 和 C_zp_dtype 的有效取值是 DT_INT8 和 DT_UINT8。

4.15.23 tpu_bdc_int8_pc_zp_mm_L_const_R_trans

左矩阵乘以右矩阵的转置，其中左矩阵的所有元素都相同，转置后的右矩阵的元素按 column 减 zero-point，结果没有 saturation。

```
void tpu_bdc_int8_pc_zp_mm_L_const_R_trans(local_addr_t output_addr,
                                             local_addr_t right_addr,
                                             local_addr_t zp_addr, scalar_t C,
                                             int left_rows, int left_cols, int
                                             right_rows, data_type_t C_dtype,
                                             data_type_t right_zp_dtype)
```

$$\text{output}_{M \times N} = \text{left}_{M \times K} \times (\text{right}_{N \times K}^T - \text{zp}_{1 \times N})$$

$$\text{output}(m, n) = \sum_k C \times (\text{right}(n, k) - \text{zp}(0, n))$$

参数

- output_addr - output 的地址
- right_addr - right 的地址
- zp_addr - zp 的地址
- C - 左矩阵元素值
- left_rows - 左矩阵的行数
- left_cols - 左矩阵的列数
- right_rows - 右矩阵的行数
- C_dtype - C 的数据类型

- `right_zp_dtype` - `right` 和 `zp` 的元素的数据类型

注意事项

- `output` 和 `right` 从 NPU 0 开始，都是 64-byte aligned layout。
- `output` 的 shape 是 `[1, left_rows, 1, right_rows]`, `left` 的 shape 是 `[1, left_rows, 1, left_cols]`, `right` 的 shape 是 `[1, right_rows, 1, left_cols]`, `zp` 的 shape 是 `[1, NPU_NUM, 1, right_rows]`, 而不是 `[1, 1, 1, right_rows]`, 每个 NPU 中都有一份相同的。
- `left_rows`、`left_cols` 和 `right_rows` 的取值范围是 `[1, 65535]`。
- `output` 的元素的数据类型是 `DT_INT32`, `C_dtype` 和 `right_zp_dtype` 的有效取值是 `DT_INT8` 和 `DT_UINT8`。

4.15.24 tpu_bdc_int8_pc_zp_mm_L_const_all_trans

两个矩阵的转置相乘，其中左矩阵的所有元素都相同，转置前的右矩阵的元素按 row 减 zero-point, 结果也转置，再对结果做累加（可选），结果没有 saturation。

```
void tpu_bdc_int8_pc_zp_mm_L_const_all_trans(local_addr_t output_addr,
                                              local_addr_t right_addr,
                                              local_addr_t zp_addr, scalar_t C,
                                              int left_rows, int left_cols, int
                                              right_rows, data_type_t C_dtype,
                                              data_type_t right_zp_dtype, bool
                                              result_add)
```

$$\text{output}_{N \times M}^T = \text{output}_{N \times M}^T + \text{left}_{K \times M}^T \times (\text{right}_{N \times K} - \text{zp}_{N \times 1})^T$$

$$\text{output}(n, m) = \text{output}(n, m) + \sum_k C \times (\text{right}(n, k) - \text{zp}(n, 0))$$

参数

- `output_addr` - `output` 的地址
- `right_addr` - `right` 的地址
- `zp_addr` - `zp` 的地址
- `C` - 左矩阵元素值
- `left_rows` - 左矩阵的行数
- `left_cols` - 左矩阵的列数
- `right_rows` - 右矩阵的行数
- `C_dtype` - `C` 的数据类型
- `right_zp_dtype` - `right` 和 `zp` 的元素的数据类型
- `result_add` - 对结果做累加的标志

注意事项

- `output`、`right` 和 `zp` 从 NPU 0 开始，`output` 和 `right` 是 64-byte aligned layout, `zp` 是 compact layout。
- `output` 的 shape 是 `[1, right_rows, 1, left_cols]`, `left` 的 shape 是 `[1, left_rows, 1, left_cols]`, `right` 的 shape 是 `[1, right_rows, 1, left_rows]`, `zp` 的 shape 是 `[1, right_rows, 1, 1]`。
- `left_rows`、`left_cols` 和 `right_rows` 的取值范围是 `[1, 65535]`。
- `output` 的元素的数据类型是 `DT_INT32`, `C_dtype` 和 `right_zp_dtype` 的有效取值是 `DT_INT8` 和 `DT_UINT8`。

4.15.25 tpu_bdc_int8_pc_zp_mm_R_const_all_trans

两个矩阵的转置相乘，其中右矩阵的所有元素都相同，转置前的右矩阵的元素按 row 减 zero-point，结果也转置，再对结果做累加（可选），结果没有 saturation。

```
void tpu_bdc_int8_pc_zp_mm_R_const_all_trans(local_addr_t output_addr,
                                              local_addr_t left_addr,
                                              local_addr_t zp_addr, scalar_t C,
                                              int left_rows, int left_cols, int
                                              right_rows, data_type_t
                                              left_dtype, data_type_t
                                              C_zp_dtype, bool result_add)
```

$$\text{output}_{N \times M}^T = \text{output}_{N \times M}^T + \text{left}_{K \times M}^T \times (\text{right}_{N \times K} - \text{zp}_{N \times 1})^T$$

$$\text{output}(n, m) = \text{output}(n, m) + \sum_k \text{left}(k, m) \times (C - \text{zp}(n, 0))$$

参数

- output_addr - output 的地址
- left_addr - left 的地址
- C - 右矩阵元素值
- zp_val - zero-point 的值
- left_rows - 左矩阵的行数
- left_cols - 左矩阵的列数
- right_rows - 右矩阵的行数
- left_dtype - left 的元素的数据类型
- C_zp_dtype - zp 的元素和 C 的数据类型
- result_add - 对结果做累加的标志

注意事项

- output、left 和 zp 从 NPU 0 开始，output 和 left 是 64-byte aligned layout，zp 是 compact layout。
- output 的 shape 是 [1, right_rows, 1, left_cols]，left 的 shape 是 [1, left_rows, 1, left_cols]，right 的 shape 是 [1, right_rows, 1, left_rows]，zp 的 shape 是 [1, right_rows, 1, 1]。
- left_rows、left_cols 和 right_rows 的取值范围是 [1, 65535]。
- output 的元素的数据类型是 DT_INT32，left_dtype 和 C_zp_dtype 的有效取值是 DT_INT8 和 DT_UINT8。

4.16 浮点神经网络操作

4.16.1 tpu_bdc_fp_bias

张量的元素按 channel 加 bias。

```
void tpu_bdc_fp_bias(local_addr_t dst_addr, local_addr_t src_addr, local_addr_t
                    bias_addr, const dim4 *shape, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(n, c, h, w) + \text{bias}(0, c, 0, 0)$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `bias_addr` -bias 的地址
- `shape` -指向 dst 和 src 的 shape 的指针
- `dtype` -dst、src 和 bias 的元素的数据类型

注意事项

- dst、src 和 bias 从同一个 NPU 开始。
- dst 和 src 是 64-byte aligned layout, bias 的 shape 是 [1, shape->c, 1, 1], compact layout。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- dtype 的有效取值是DT_FP32、DT_FP16 和DT_BFP16。

4.16.2 tpu_bdc_fp_scale

张量的元素按 channel 乘 scale。

```
void tpu_bdc_fp_scale(local_addr_t dst_addr, local_addr_t src_addr, local_addr_t
                      scale_addr, const dim4 *shape, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(n, c, h, w) \times \text{scale}(0, c, 0, 0)$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `scale_addr` -scale 的地址
- `shape` -指向 dst 和 src 的 shape 的指针
- `dtype` -dst、src 和 scale 的元素的数据类型

注意事项

- dst、src 和 scale 从同一个 NPU 开始。
- dst 和 src 是 64-byte aligned layout, scale 的 shape 是 [1, shape->c, 1, 1], compact layout。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- dtype 的有效取值是DT_FP32、DT_FP16 和DT_BFP16。

4.16.3 tpu_bdc_fp_scale_bias

张量的元素按 channel 乘 scale, 再加 bias。

```
void tpu_bdc_fp_scale_bias(local_addr_t dst_addr, local_addr_t src_addr,
                           local_addr_t scale_addr, local_addr_t bias_addr, const
                           dim4 *shape, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(n, c, h, w) \times \text{scale}(0, c, 0, 0) + \text{bias}(0, c, 0, 0)$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `scale_addr` -scale 的地址
- `bias_addr` -bias 的地址
- `shape` -指向 dst 和 src 的 shape 的指针
- `dtype` -dst、src、scale 和 bias 的元素的数据类型

注意事项

- dst、src、scale 和 bias 从同一个 NPU 开始。
- dst 和 src 是 64-byte aligned layout, scale 和 bias 的 shape 是 [1, shape->c, 1, 1], compact layout。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- dtype 的有效取值是DT_FP32、DT_FP16 和DT_BFP16。

4.16.4 tpu_bdc_fp_scale_bias_C

张量的元素按 channel 乘 scale, 再加 bias。

```
void tpu_bdc_fp_scale_bias_C(local_addr_t dst_addr, local_addr_t src_addr,
                             scalar_t scale, scalar_t bias, const dim4 *shape,
                             data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \text{src}(n, c, h, w) \times \text{scale} + \text{bias}$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `scale` -scale 常数
- `bias` -bias 常数
- `shape` -指向 dst 和 src 的 shape 的指针
- `dtype` -dst 和 src 的元素和 scale 和 bias 的数据类型

注意事项

- dst 和 src 从同一个 NPU 开始, 都是 64-byte aligned layout。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- dtype 的有效取值是DT_FP32、DT_FP16 和DT_BFP16。

4.16.5 tpu_bdc_fp_add_bias_sqr

张量的元素按 channel 加 bias, 结果再平方。

```
void tpu_bdc_fp_add_bias_sqr(local_addr_t dst_addr, local_addr_t src_addr,
                             local_addr_t bias_addr, const dim4 *shape,
                             data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = (\text{src}(n, c, h, w) + \text{bias}(0, c, 0, 0))^2$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `bias_addr` -bias 的地址
- `shape` -指向 dst 和 src 的 shape 的指针
- `dtype` -dst、src 和 bias 的元素的数据类型

注意事项

- dst、src 和 bias 从同一个 NPU 开始。
- dst 和 src 是 64-byte aligned layout, bias 的 shape 是 [1, shape->c, 1, 1], compact layout。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- dtype 的有效取值是DT_FP32、DT_FP16 和DT_BFP16。

4.16.6 tpu_bdc_fp_add_C_sqr

张量的元素加常数，结果再平方。

```
void tpu_bdc_fp_add_C_sqr(local_addr_t dst_addr, local_addr_t src_addr, scalar_t
                          C, const dim4 *shape, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = (\text{src}(n, c, h, w) + C)^2$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `C` -常数
- `shape` -指向 dst 和 src 的 shape 的指针
- `dtype` -dst 和 src 的元素和 C 的数据类型

注意事项

- dst 和 src 从同一个 NPU 开始，都是 64-byte aligned layout。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- dtype 的有效取值是DT_FP32、DT_FP16 和DT_BFP16。

4.16.7 tpu_bdc_fp_sub_bias_sqr

张量的元素按 channel 减 bias，结果再平方。

```
void tpu_bdc_fp_sub_bias_sqr(local_addr_t dst_addr, local_addr_t src_addr,
                              local_addr_t bias_addr, const dim4 *shape,
                              data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = (\text{src}(n, c, h, w) - \text{bias}(0, c, 0, 0))^2$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址

- `bias_addr` - bias 的地址
- `shape` - 指向 `dst` 和 `src` 的 `shape` 的指针
- `dtype` - `dst`、`src` 和 `bias` 的元素的数据类型

注意事项

- `dst`、`src` 和 `bias` 从同一个 NPU 开始。
- `dst` 和 `src` 是 64-byte aligned layout, `bias` 的 `shape` 是 `[1, shape->c, 1, 1]`, compact layout。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 `[1, 65535]`。
- `dtype` 的有效取值是 `DT_FP32`、`DT_FP16` 和 `DT_BFP16`。

4.16.8 tpu_bdc_fp_sub_C_sqr

张量的元素减常数，结果再平方。

```
void tpu_bdc_fp_sub_C_sqr(local_addr_t dst_addr, local_addr_t src_addr, scalar_t
                          C, const dim4 *shape, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = (\text{src}(n, c, h, w) - C)^2$$

参数

- `dst_addr` - `dst` 的地址
- `src_addr` - `src` 的地址
- `C` - 常数
- `shape` - 指向 `dst` 和 `src` 的 `shape` 的指针
- `dtype` - `dst` 和 `src` 的元素和 `C` 的数据类型

注意事项

- `dst` 和 `src` 从同一个 NPU 开始，都是 64-byte aligned layout。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 `[1, 65535]`。
- `dtype` 的有效取值是 `DT_FP32`、`DT_FP16` 和 `DT_BFP16`。

4.16.9 tpu_bdc_fp_conv2d

2D 卷积，结果按 channel 加 bias（可选），再对结果做累加（可选）。

```
void tpu_bdc_fp_conv2d(local_addr_t output_addr, local_addr_t input_addr,
                       local_addr_t weight_addr, local_addr_t bias_addr, const
                       dim4 *input_shape, const dim4 *input_stride, int output_c,
                       const dim2 *kernel, const padding_t *padding, const dim2
                       *stride, const dim2 *dilation, data_type_t output_dtype,
                       data_type_t input_dtype, bool has_bias, bool result_add)
```

参数

- `output_addr` - output 的地址
- `input_addr` - input 的地址
- `weight_addr` - weight 的地址
- `bias_addr` - bias 的地址

- `input_shape` - 指向 `input` 的 `shape` 的指针
- `input_stride` - 指向 `input` 的 `stride` 的指针
- `output_c` - `output` 的 `channel` 数
- `kernel` - 指向 `kernel` 的 `size` 的指针
- `padding` - 指向 `padding` 的 `size` 指针
- `stride` - 指向 `stride` 的指针
- `dilation` - 指向 `dilation` 的指针
- `output_dtype` - `output` 的元素的 `数据类型`
- `input_dtype` - 仅代表 `input` 和 `weight` 元素的 `数据类型`; 如果有 `bias`, 那么本 API 函数接收的 `bias_addr` 参数地址处的 `bias` 元素的默认数据存储类型为 `DT_FP32`(即 `float`)
- `has_bias` - 加 `bias` 的标志
- `result_add` - 对结果做累加的标志

注意事项

- `output`、`weight` 和 `bias` 从同一个 NPU 开始, `input` 从 NPU 0 开始。
- `output` 的 `shape` 是 `[input_shape->n, output_c, output_h, output_w]`, 64-byte aligned layout,

$$\text{output_h} = (\text{input_shape->h} + \text{padding->top} + \text{padding->bottom} - ((\text{kernel->h} - 1) * \text{dilation->h} + 1)) / \text{stride->h} + 1,$$

$$\text{output_w} = (\text{input_shape->w} + \text{padding->left} + \text{padding->right} - ((\text{kernel->w} - 1) * \text{dilation->w} + 1)) / \text{stride->w} + 1.$$
- `input` 是 `free layout`, `bias` 的 `shape` 是 `[1, output_c, 1, 1]`, `compact layout`, `weight` 的 `shape` 是 `[input_shape->c, output_c, kernel->h, kernel->w]`, 如果 `input_dtype` 是 `DT_FP32`, 则 `weight` 是 `compact layout`, 否则是 `32-IC layout`。
- `input_shape->n`、`input_shape->c`、`input_shape->h`、`input_shape->w`、`kernel->h` 和 `kernel->w` 的取值范围是 `[1, 65535]`。
- `input_dtype` 的有效取值是 `DT_FP32`、`DT_FP16` 和 `DT_BFP16`, `output_dtype` 与 `input_dtype` 相同, 或是 `DT_FP32`。
- `padding->top`、`padding->bottom`、`padding->left` 和 `padding->right` 的取值范围是 `[0, 15]`, `stride->h`、`stride->w`、`dilation->h` 和 `dilation->w` 的取值范围是 `[1, 15]`。

4.16.10 tpu_bdc_fp_conv2d_kernel_const

核为常数的 2D 卷积, 结果按 `channel` 加 `bias` (可选), 再对结果做累加 (可选)。

```
void tpu_bdc_fp_conv2d_kernel_const(local_addr_t output_addr, local_addr_t
                                     input_addr, local_addr_t bias_addr, scalar_t
                                     C, const dim4 *input_shape, const dim4
                                     *input_stride, int output_c, const dim2 *kernel,
                                     const padding_t *padding, const dim2 *stride,
                                     const dim2 *dilation, data_type_t
                                     output_dtype, data_type_t input_dtype, bool
                                     has_bias, bool result_add)
```

参数

- `output_addr` - `output` 的地址
- `input_addr` - `input` 的地址

- `bias_addr` - bias 的地址
- `C_weight` 的常数值
- `input_shape` - 指向 input 的 shape 的指针
- `input_stride` - 指向 input 的 stride 的指针
- `output_c` - output 的 channel 数
- `kernel` - 指向 kernel 的 size 的指针
- `padding` - 指向 padding 的 size 指针
- `stride` - 指向 stride 的指针
- `dilation` - 指向 dilation 的指针
- `output_dtype` - output 的元素的数据类型
- `input_dtype` - 仅代表 input 和 weight 元素的数据类型; 如果有 bias, 那么本 API 函数接收的 `bias_addr` 参数地址处的 bias 元素的默认数据存储类型为 `DT_FP32`(即 float)
- `has_bias` - 加 bias 的标志
- `result_add` - 对结果做累加的标志

注意事项

- output、weight 和 bias 从同一个 NPU 开始, input 从 NPU 0 开始。
- output 的 shape 是 `[input_shape->n, output_c, output_h, output_w]`, 64-byte aligned layout,

$$\text{output_h} = (\text{input_shape->h} + \text{padding->top} + \text{padding->bottom} - ((\text{kernel->h} - 1) * \text{dilation->h} + 1)) / \text{stride->h} + 1,$$

$$\text{output_w} = (\text{input_shape->w} + \text{padding->left} + \text{padding->right} - ((\text{kernel->w} - 1) * \text{dilation->w} + 1)) / \text{stride->w} + 1.$$
- input 是 free layout, bias 的 shape 是 `[1, output_c, 1, 1]`, compact layout, weight 的 shape 是 `[input_shape->c, output_c, kernel->h, kernel->w]`,
- `input_shape->n`、`input_shape->c`、`input_shape->h`、`input_shape->w`、`kernel->h` 和 `kernel->w` 的取值范围是 `[1, 65535]`。
- `input_dtype` 的有效取值是 `DT_FP32`、`DT_FP16` 和 `DT_BFP16`, `output_dtype` 与 `input_dtype` 相同, 或是 `DT_FP32`。
- `padding->top`、`padding->bottom`、`padding->left` 和 `padding->right` 的取值范围是 `[0, 15]`, `stride->h`、`stride->w`、`dilation->h` 和 `dilation->w` 的取值范围是 `[1, 15]`。

4.16.11 tpu_bdc_fp_conv2d_for_deconv2d

做 2D 反卷积的 2D 卷积 (input 可以插零, kernel 做旋转), 结果按 channel 加 bias (可选), 再对结果做累加 (可选)。

```
void tpu_bdc_fp_conv2d_for_deconv2d(local_addr_t output_addr, local_addr_t
                                     input_addr, local_addr_t weight_addr,
                                     local_addr_t bias_addr, const dim4
                                     *input_shape, const dim4 *input_stride, int
                                     output_c, const dim2 *kernel, const dim2
                                     *insert, const padding_t *padding, const dim2
                                     *dilation, data_type_t output_dtype,
                                     data_type_t input_dtype, bool has_bias, bool
                                     result_add)
```

参数

- `output_addr` - output 的地址
- `input_addr` - input 的地址
- `weight_addr` - weight 的地址
- `bias_addr` - bias 的地址
- `input_shape` - 指向 input 的 shape 的指针
- `input_stride` - 指向 input 的 stride 的指针
- `output_c` - output 的 channel 数
- `kernel` - 指向 kernel 的 size 的指针
- `insert` - 指向 insert 的指针
- `padding` - 指向 padding 的 size 指针
- `dilation` - 指向 dilation 的指针
- `output_dtype` - output 的元素的数据类型
- `input_dtype` - 仅代表 input 和 weight 元素的数据类型; 如果有 bias, 那么本 API 函数接收的 `bias_addr` 参数地址处的 bias 元素的默认数据存储类型为 `DT_FP32`(即 float)
- `has_bias` - 加 bias 的标志
- `result_add` - 对结果做累加的标志

注意事项

- output、weight 和 bias 从同一个 NPU 开始, input 从 NPU 0 开始。
- output 的 shape 是 `[input_shape->n, output_c, output_h, output_w]`, 64-byte aligned layout,

$$\text{output_h} = ((\text{input_shape->h} - 1) * (\text{insert->h} + 1) + 1 + \text{padding->top} + \text{padding->bottom} - ((\text{kernel->h} - 1) * \text{dilation->h} + 1)) / \text{stride->h} + 1,$$

$$\text{output_w} = ((\text{input_shape->w} - 1) * (\text{insert->w} + 1) + 1 + \text{padding->left} + \text{padding->right} - ((\text{kernel->w} - 1) * \text{dilation->w} + 1)) / \text{stride->w} + 1.$$
- input 是 free layout, bias 的 shape 是 `[1, output_c, 1, 1]`, compact layout, weight 的 shape 是 `[input_shape->c, output_c, kernel->h, kernel->w]`, 如果 `input_dtype` 是 `DT_FP32`, 则 weight 是 compact layout, 否则是 32-IC layout。
- `input_shape->n`、`input_shape->c`、`input_shape->h`、`input_shape->w`、`kernel->h` 和 `kernel->w` 的取值范围是 `[1, 65535]`。
- `input_dtype` 的有效取值是 `DT_FP32`、`DT_FP16` 和 `DT_BFP16`, `output_dtype` 与 `input_dtype` 相同, 或是 `DT_FP32`。
- `padding->top`、`padding->bottom`、`padding->left` 和 `padding->right` 的取值范围是 `[0, 15]`, `insert->h` 和 `insert->w` 的取值范围是 `[0, 14]`, `dilation->h` 和 `dilation->w` 的取值范围是 `[1, 15]`。

4.16.12 tpu_bdc_fp_max_pool2d

2D 最大池化，可自定义 padding 的值。

```
void tpu_bdc_fp_max_pool2d(local_addr_t output_addr, local_addr_t input_addr,
                           const dim4 *input_shape, const dim2 *kernel, const
                           padding_t *padding, const dim2 *stride, const dim2
                           *dilation, data_type_t dtype, scalar_t pad_val)
```

参数

- output_addr - output 的地址
- input_addr - input 的地址
- input_shape - 指向 input 的 shape 的指针
- kernel - 指向 kernel 的 size 的指针
- padding - 指向 padding 的 size 指针
- stride - 指向 stride 的指针
- dilation - 指向 dilation 的指针
- dtype - output 和 input 的元素和 pad_val 的数据类型
- pad_val - padding 的值

注意事项

- output 和 input 从同一个 NPU 开始，都是 64-byte aligned layout。
- output 的 shape 是 [input_shape->n, input_shape->c, output_h, output_w],

$$\text{output_h} = (\text{input_shape->h} + \text{padding->top} + \text{padding->bottom} - ((\text{kernel->h} - 1) * \text{dilation->h} + 1)) / \text{stride->h} + 1,$$

$$\text{output_w} = (\text{input_shape->w} + \text{padding->left} + \text{padding->right} - ((\text{kernel->w} - 1) * \text{dilation->w} + 1)) / \text{stride->w} + 1.$$
- input_shape->n、input_shape->c、input_shape->h、input_shape->w、kernel->h 和 kernel->w 的取值范围是 [1, 65535]。
- dtype 的有效取值是 DT_FP32、DT_FP16 和 DT_BFP16。
- padding->top、padding->bottom、padding->left 和 padding->right 的取值范围是 [0, 15]，stride->h、stride->w、dilation->h 和 dilation->w 的取值范围是 [1, 15]。

4.16.13 tpu_bdc_fp_ins_avg_pool2d

2D 均值池化，可自定义均值 scale 值来代替传统的 $1 / (\text{kernel->h} * \text{kernel->w})$ 。

```
void tpu_bdc_fp_ins_avg_pool2d(local_addr_t output_addr, local_addr_t input_addr,
                                const dim4 *input_shape, const dim2 *kernel, const
                                padding_t *padding, const dim2 *stride, const dim2
                                *dilation, const dim2 *ins, data_type_t dtype,
                                scalar_t scale)
```

参数

- output_addr - output 的地址
- input_addr - input 的地址
- input_shape - 指向 input 的 shape 的指针

- `kernel` –指向 `kernel` 的 `size` 的指针
- `padding` –指向 `padding` 的指针
- `stride` –指向 `stride` 的指针
- `dilation` –指向 `dilation` 的指针
- `ins` –指向 `ins` 的指针，对输入 `feature map` 的行/列之间插入元素的个数
- `dtype` –output 和 input 的元素和 `scale` 的数据类型
- `scale` –均值 `scale` 值

注意事项

- output 和 input 从同一个 NPU 开始，都是 64-byte aligned layout。
- output 的 shape 是 `[input_shape->n, input_shape->c, output_h, output_w]`,

$$\text{output_h} = (\text{input_shape->h} + \text{padding->top} + \text{padding->bottom} - ((\text{kernel->h} - 1) * \text{dilation->h} + 1)) / \text{stride->h} + 1,$$

$$\text{output_w} = (\text{input_shape->w} + \text{padding->left} + \text{padding->right} - ((\text{kernel->w} - 1) * \text{dilation->w} + 1)) / \text{stride->w} + 1.$$
- `input_shape->n`、`input_shape->c`、`input_shape->h`、`input_shape->w`、`kernel->h` 和 `kernel->w` 的取值范围是 `[1, 65535]`。
- `dtype` 的有效取值是 `DT_FP32`、`DT_FP16` 和 `DT_BFP16`。
- `padding->top`、`padding->bottom`、`padding->left` 和 `padding->right` 的取值范围是 `[0, 15]`，`stride->h`、`stride->w`、`dilation->h` 和 `dilation->w` 的取值范围是 `[1, 15]`。
- `ins->h`、`ins->w` 的取值范围是 `[0, 8)`。

4.16.14 tpu_bdc_fp_avg_pool2d

2D 均值池化，可自定义均值 `scale` 值来代替传统的 $1 / (\text{kernel->h} * \text{kernel->w})$ 。

```
void tpu_bdc_fp_avg_pool2d(local_addr_t output_addr, local_addr_t input_addr,
                           const dim4 *input_shape, const dim2 *kernel, const
                           padding_t *padding, const dim2 *stride, const dim2
                           *dilation, data_type_t dtype, scalar_t scale)
```

参数

- `output_addr` –output 的地址
- `input_addr` –input 的地址
- `input_shape` –指向 input 的 shape 的指针
- `kernel` –指向 `kernel` 的 `size` 的指针
- `padding` –指向 `padding` 的指针
- `stride` –指向 `stride` 的指针
- `dilation` –指向 `dilation` 的指针
- `dtype` –output 和 input 的元素和 `scale` 的数据类型
- `scale` –均值 `scale` 值

注意事项

- output 和 input 从同一个 NPU 开始，都是 64-byte aligned layout。

- output 的 shape 是 [input_shape->n, input_shape->c, output_h, output_w],

$$\text{output_h} = (\text{input_shape->h} + \text{padding->top} + \text{padding->bottom} - ((\text{kernel->h} - 1) * \text{dilation->h} + 1)) / \text{stride->h} + 1,$$

$$\text{output_w} = (\text{input_shape->w} + \text{padding->left} + \text{padding->right} - ((\text{kernel->w} - 1) * \text{dilation->w} + 1)) / \text{stride->w} + 1.$$
- input_shape->n、input_shape->c、input_shape->h、input_shape->w、kernel->h 和 kernel->w 的取值范围是 [1, 65535]。
- dtype 的有效取值是DT_FP32、DT_FP16 和DT_BFP16。
- padding->top、padding->bottom、padding->left 和 padding->right 的取值范围是 [0, 15], stride->h、stride->w、dilation->h 和 dilation->w 的取值范围是 [1, 15]。

4.16.15 tpu_bdc_fp_depthwise2d

2D depthwise 卷积，结果按 channel 加 bias（可选），再对结果做累加（可选）。

```
void tpu_bdc_fp_depthwise2d(local_addr_t output_addr, local_addr_t input_addr,
                           local_addr_t weight_addr, local_addr_t bias_addr,
                           const dim4 *input_shape, const dim2 *kernel, const
                           padding_t *padding, const dim2 *stride, const dim2
                           *dilation, data_type_t dtype, bool has_bias)
```

参数

- output_addr - output 的地址
- input_addr - input 的地址
- weight_addr - weight 的地址
- bias_addr - bias 的地址
- input_shape - 指向 input 的 shape 的指针
- kernel - 指向 kernel 的 size 的指针
- padding - 指向 padding 的指针
- stride - 指向 stride 的指针
- dilation - 指向 dilation 的指针
- dtype - output、input、weight 和 bias 的元素的数据类型
- has_bias - 加 bias 的标志

注意事项

- output、input、weight 和 bias 从同一个 NPU 开始。
- output 的 shape 是 [input_shape->n, input_shape->c, output_h, output_w], 64-byte aligned layout,

$$\text{output_h} = (\text{input_shape->h} + \text{padding->top} + \text{padding->bottom} - ((\text{kernel->h} - 1) * \text{dilation->h} + 1)) / \text{stride->h} + 1,$$

$$\text{output_w} = (\text{input_shape->w} + \text{padding->left} + \text{padding->right} - ((\text{kernel->w} - 1) * \text{dilation->w} + 1)) / \text{stride->w} + 1.$$
- input 是 64-byte aligned layout, weight 的 shape 是 [1, input_shape->c, kernel->h, kernel->w], compact layout, bias 的 shape 是 [1, input_shape->c, 1, 1], compact layout。
- input_shape->n、input_shape->c、input_shape->h、input_shape->w、kernel->h 和 kernel->w 的取值范围是 [1, 65535]。

- dtype 的有效取值是DT_FP32、DT_FP16 和DT_BFP16。
- padding->top、padding->bottom、padding->left 和 padding->right 的取值范围是 [0, 15], stride->h、stride->w、dilation->h 和 dilation->w 的取值范围是 [1, 15]。

4.17 定点神经网络操作

4.17.1 tpu_bdc_int8_max_pool2d

2D 最大池化，可自定义 padding 的值。

```
void tpu_bdc_int8_max_pool2d(local_addr_t output_addr, local_addr_t input_addr,
                             const dim4 *input_shape, const dim2 *kernel, const
                             padding_t *padding, const dim2 *stride, const dim2
                             *dilation, data_type_t dtype, scalar_t pad_val)
```

参数

- output_addr -output 的地址
- input_addr -input 的地址
- input_shape -指向 input 的 shape 的指针
- kernel -指向 kernel 的 size 的指针
- padding -指向 padding 的 size 指针
- stride -指向 stride 的指针
- dilation -指向 dilation 的指针
- dtype -output 和 input 的元素和 pad_val 的数据类型
- pad_val -padding 的值

注意事项

- output 和 input 从同一个 NPU 开始，都是 64-byte aligned layout。
- output 的 shape 是 [input_shape->n, input_shape->c, output_h, output_w],

$$\text{output_h} = (\text{input_shape->h} + \text{padding->top} + \text{padding->bottom} - ((\text{kernel->h} - 1) * \text{dilation->h} + 1)) / \text{stride->h} + 1,$$

$$\text{output_w} = (\text{input_shape->w} + \text{padding->left} + \text{padding->right} - ((\text{kernel->w} - 1) * \text{dilation->w} + 1)) / \text{stride->w} + 1.$$
- input_shape->n、input_shape->c、input_shape->h、input_shape->w、kernel->h 和 kernel->w 的取值范围是 [1, 65535]。
- dtype 的有效取值是DT_INT8 和DT_UINT8。
- padding->top、padding->bottom、padding->left 和 padding->right 的取值范围是 [0, 15], stride->h、stride->w、dilation->h 和 dilation->w 的取值范围是 [1, 15]。

4.17.2 tpu_bdc_int8_avg_pool2d

2D 均值池化，可自定义均值 scale 值，对结果做算术移位，结果有 saturation。

```
void tpu_bdc_int8_avg_pool2d(local_addr_t output_addr, local_addr_t input_addr,
                             const dim4_t *input_shape, const dim2_t *kernel, const
                             padding_t *padding, const dim2_t *stride, const dim2_t
                             *dilation, data_type_t output_dtype, data_type_t
                             input_dtype, unsigned char scale, unsigned char rshift)
```

参数

- output_addr - output 的地址
- input_addr - input 的地址
- input_shape - 指向 input 的 shape 的指针
- kernel - 指向 kernel 的 size 的指针
- padding - 指向 padding 的指针
- stride - 指向 stride 的指针
- dilation - 指向 dilation 的指针
- output_dtype - output 的元素的数据类型
- input_dtype - input 的元素和 scale 的数据类型
- scale - 均值 scale 值
- rshift - 移位数

注意事项

- output 和 input 从同一个 NPU 开始，都是 64-byte aligned layout。
- output 的 shape 是 [input_shape->n, input_shape->c, output_h, output_w],

$$\text{output_h} = (\text{input_shape->h} + \text{padding->top} + \text{padding->bottom} - ((\text{kernel->h} - 1) * \text{dilation->h} + 1)) / \text{stride->h} + 1,$$

$$\text{output_w} = (\text{input_shape->w} + \text{padding->left} + \text{padding->right} - ((\text{kernel->w} - 1) * \text{dilation->w} + 1)) / \text{stride->w} + 1.$$
- input_shape->n、input_shape->c、input_shape->h、input_shape->w、kernel->h 和 kernel->w 的取值范围是 [1, 65535]。
- output_dtype 的有效取值是 DT_INT32、DT_UINT32、DT_INT16、DT_UINT16、DT_INT8 和 DT_UINT8, input_dtype 的有效取值是 DT_INT8 和 DT_UINT8, output_dtype 和 input_dtype 的符号相同。
- padding->top、padding->bottom、padding->left 和 padding->right 的取值范围是 [0, 15], stride->h、stride->w、dilation->h 和 dilation->w 的取值范围是 [1, 15]。
- rshift 的取值范围是 [0, 31]。

4.18 激活函数

4.18.1 tpu_bdc_relu

张量的元素做 ReLU 操作。

```
void tpu_bdc_relu(local_addr_t dst_addr, local_addr_t src_addr, const dim4 *shape,
                  const dim4 *dst_stride, const dim4 *src_stride, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{src}(n, c, h, w) & \text{如果 } \text{shift}(n, c, h, w) > 0 \\ 0 & \text{其他情况} \end{cases}$$

参数

- dst_addr –dst 的地址
- src_addr –src 的地址
- shape –指向 dst 和 src 的 shape 的指针
- dst_stride –指向 dst 的 stride 的指针
- src_stride –指向 src 的 stride 的指针
- dtype –dst 和 src 的元素的数据类型

注意事项

- dst 和 src 从同一个 NPU 开始。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- 如果张量的 stride 指针是 NULL，则此张量是 64-byte aligned layout，否则是 free layout。

4.18.2 tpu_bdc_prelu

张量的元素做 PReLU 操作。

```
void tpu_bdc_prelu(local_addr_t dst_addr, local_addr_t src_addr, scalar_t alpha,
                   const dim4 *shape, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{src}(n, c, h, w) & \text{如果 } \text{shift}(n, c, h, w) > 0 \\ \alpha \times \text{src}(n, c, h, w) & \text{其他情况} \end{cases}$$

参数

- dst_addr –dst 的地址
- src_addr –src 的地址
- alpha –常数
- shape –指向 dst 和 src 的 shape 的指针
- dtype –dst 和 src 的元素和 alpha 的数据类型

注意事项

- dst 和 src 从同一个 NPU 开始，都是 64-byte aligned layout。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。

4.18.3 tpu_bdc_fp32_elu

张量的元素做 ELU 操作。

```
void tpu_bdc_fp32_elu(local_addr_t dst_addr, local_addr_t src_addr, local_addr_t
                      work0_addr, local_addr_t work1_addr, local_addr_t
                      coeff_addr, local_addr_t table_addr, float alpha, const dim4
                      *shape)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{src}(n, c, h, w) & \text{如果 } \text{shift}(n, c, h, w) > 0 \\ \alpha \times (e^{\text{src}(n, c, h, w)} - 1) & \text{其他情况} \end{cases}$$

参数

- dst_addr -dst 的地址
- src_addr -src 的地址
- work0_addr -work0 的地址
- work1_addr -work1 的地址
- coeff_addr -coeff 的地址
- table_addr -table 的地址
- alpha -常数
- shape -dst、src、work0 和 work1 的 shape 的指针

注意事项

- dst、src、work0 和 work1 从同一个 NPU 开始，都是 64-byte aligned layout。
- coeff 通过 `tpu_bdc_load_fp32_exp_coeff()` 加载，table 通过 `tpu_bdc_load_fp32_exp_table()` 加载。
- work0 和 work1 存放中间结果，不允许 dst_addr、src_addr、work0_addr 和 work1_addr 任意两个相等，如果 dst、work1 和 table 两两之间没有 bank conflict-ing，则性能更优。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]，shape->h * shape->w 小于等于 65535。
- 如果 src(n, c, h, w) 小于 -103 或大于 88，则 $e^{\text{src}(n, c, h, w)}$ 分别是 e^{-103} 和 e^{88} 。

4.18.4 tpu_bdc_fp32_sigmoid

张量的元素做 sigmoid 操作。

```
void tpu_bdc_fp32_sigmoid(local_addr_t dst_addr, local_addr_t src_addr,
                          local_addr_t work0_addr, local_addr_t work1_addr,
                          local_addr_t coeff_addr, local_addr_t table_addr, const
                          dim4 *shape)
```

$$\text{dst}(n, c, h, w) = \frac{1}{1 + e^{-\text{src}(n, c, h, w)}}$$

参数

- dst_addr -dst 的地址
- src_addr -src 的地址
- work0_addr -work0 的地址

- `work1_addr` - `work1` 的地址
- `coeff_addr` - `coeff` 的地址
- `table_addr` - `table` 的地址
- `shape` - `dst`、`src`、`work0` 和 `work1` 的 `shape` 的指针

注意事项

- `dst`、`src`、`work0` 和 `work1` 从同一个 NPU 开始，都是 64-byte aligned layout。
- `coeff` 通过 `tpu_bdc_load_fp32_exp_coeff()` 加载，`table` 通过 `tpu_bdc_load_fp32_exp_table()` 加载。
- `work0` 和 `work1` 存放中间结果，不允许 `dst_addr`、`work0_addr` 和 `work1_addr` 任意两个相等，允许 `src_addr` 等于 `dst_addr`、`work0_addr` 或 `work1_addr`，此情况下 `src` 的数据会被覆盖，如果 `dst`、`work1` 和 `table` 两两之间没有 bank conflicting，则性能更优。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]，`shape->h * shape->w` 小于等于 65535。
- 如果 `src(n, c, h, w)` 小于 -88 或大于 103，则 $e^{-src(n, c, h, w)}$ 分别是 e^{88} 和 e^{-103} 。

4.18.5 tpu_bdc_fp32_tanh

张量的元素做 tanh 操作。

```
void tpu_bdc_fp32_tanh(local_addr_t dst_addr, local_addr_t src_addr, local_addr_t
                      work0_addr, local_addr_t work1_addr, local_addr_t
                      coeff_addr, local_addr_t table_addr, const dim4 *shape)
```

$$dst(n, c, h, w) = \frac{e^{src(n, c, h, w)} - e^{-src(n, c, h, w)}}{e^{src(n, c, h, w)} + e^{-src(n, c, h, w)}}$$

参数

- `dst_addr` - `dst` 的地址
- `src_addr` - `src` 的地址
- `work0_addr` - `work0` 的地址
- `work1_addr` - `work1` 的地址
- `coeff_addr` - `coeff` 的地址
- `table_addr` - `table` 的地址
- `shape` - `dst`、`src`、`work0` 和 `work1` 的 `shape` 的指针

注意事项

- `dst`、`src`、`work0` 和 `work1` 从同一个 NPU 开始，都是 64-byte aligned layout。
- `coeff` 通过 `tpu_bdc_load_fp32_exp_coeff()` 加载，`table` 通过 `tpu_bdc_load_fp32_exp_table()` 加载。
- `work0` 和 `work1` 存放中间结果，不允许 `dst_addr`、`work0_addr` 和 `work1_addr` 任意两个相等，允许 `src_addr` 等于 `dst_addr`、`work0_addr` 或 `work1_addr`，此情况下 `src` 的数据会被覆盖，如果 `dst`、`work1` 和 `table` 两两之间没有 bank conflicting，则性能更优。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]，`shape->h * shape->w` 小于等于 65535。
- 如果 `src(n, c, h, w)` 小于 -103 或大于 88，则 $e^{src(n, c, h, w)}$ 分别是 e^{-103} 和 e^{88} 。如果 `src(n, c, h, w)` 小于 -88 或大于 103，则 $e^{-src(n, c, h, w)}$ 分别是 e^{88} 和 e^{-103} 。

4.18.6 tpu_bdc_fp32_softplus

张量的元素做 softplus 操作。

```
void tpu_bdc_fp32_softplus(local_addr_t dst_addr, local_addr_t src_addr,
                           local_addr_t work0_addr, local_addr_t work1_addr,
                           local_addr_t exp_coeff_addr, local_addr_t
                           log_coeff_addr, local_addr_t exp_table_addr, const dim4
                           *shape, float beta)
```

$$\text{dst}(n, c, h, w) = \frac{1}{\beta} \log(1 + e^{\beta \times \text{src}(n, c, h, w)})$$

参数

- dst_addr -dst 的地址
- src_addr -src 的地址
- work0_addr -work0 的地址
- work1_addr -work1 的地址
- exp_coeff_addr -exp_coeff 的地址
- log_coeff_addr -log_coeff 的地址
- exp_table_addr -exp_table 的地址
- shape -指向 dst、src、work0 和 work1 的 shape 的指针

注意事项

- dst、src、work0 和 work1 从同一个 NPU 开始，都是 64-byte aligned layout。
- exp_coeff 通过 tpu_bdc_load_fp32_exp_coeff() 加载，log_coeff 通过 tpu_bdc_load_fp32_log_coeff() 加载，exp_table 通过 tpu_bdc_load_fp32_exp_table() 加载。
- work0 和 work1 存放中间结果，不允许 dst_addr、work0_addr 和 work1_addr 任意两个相等，允许 src_addr 等于 dst_addr、work0_addr 或 work1_addr，此情况下 src 的数据会被覆盖，如果 dst、work1 和 table 两两之间没有 bank conflicting，则性能更优。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]，shape->h * shape->w 小于等于 65535。
- 如果 $\beta \times \text{src}(n, c, h, w)$ 小于 -103 或大于 88，则 $e^{\beta \times \text{src}(n, c, h, w)}$ 分别是 e^{-103} 和 e^{88} 。

4.18.7 tpu_bdc_fp32_erf

张量的元素做 erf 操作。

```
void tpu_bdc_fp32_erf(local_addr_t dst_addr, local_addr_t src_addr, local_addr_t
                      work0_addr, local_addr_t work1_addr, local_addr_t
                      work2_addr, local_addr_t exp_coeff_addr, local_addr_t
                      erf_coeff_addr, local_addr_t exp_table_addr, const dim4
                      *shape)
```

$$\text{dst}(n, c, h, w) = \frac{2}{\sqrt{\pi}} \int_0^{\text{src}(n, c, h, w)} e^{-t^2} dt$$

参数

- dst_addr -dst 的地址
- src_addr -src 的地址

- `work0_addr` - `work0` 的地址
- `work1_addr` - `work1` 的地址
- `work2_addr` - `work2` 的地址
- `exp_coeff_addr` - `exp_coeff` 的地址
- `erf_coeff_addr` - `erf_coeff` 的地址
- `exp_table_addr` - `exp_table` 的地址
- `shape` - 指向 `dst`、`src`、`work0`、`work1` 和 `work2` 的 `shape` 的指针

注意事项

- `dst`、`src`、`work0`、`work1` 和 `work2` 从同一个 NPU 开始，都是 64-byte aligned layout。
- `exp_coeff` 通过 `tpu_bdc_load_fp32_exp_coeff()` 加载，`erf_coeff` 通过 `tpu_bdc_load_fp32_erf_coeff()` 加载，`exp_table` 通过 `tpu_bdc_load_fp32_exp_table()` 加载。
- `work0`、`work1` 和 `work2` 存放中间结果，不允许 `dst_addr`、`src_addr`、`work0_addr`、`work1_addr` 和 `work2_addr` 任意两个相等，如果 `dst`、`work1` 和 `exp_table` 两两之间没有 bank conflicting，则性能更优。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]，`shape->h * shape->w` 小于等于 65535。
- 如果计算过程中 e^t 的指数 t 小于 -103 或大于 88，则 e^t 分别是 e^{-103} 和 e^{88} 。

4.18.8 tpu_bdc_fp32_erfc

张量的元素做 `erfc` 操作。

```
void tpu_bdc_fp32_erfc(local_addr_t dst_addr, local_addr_t src_addr, local_addr_t
                      work0_addr, local_addr_t work1_addr, local_addr_t
                      work2_addr, local_addr_t exp_coeff_addr, local_addr_t
                      erf_coeff_addr, local_addr_t exp_table_addr, const dim4
                      *shape)
```

$$\text{dst}(n, c, h, w) = 1 - \frac{2}{\sqrt{\pi}} \int_0^{\text{src}(n, c, h, w)} e^{-t^2} dt$$

参数

- `dst_addr` - `dst` 的地址
- `src_addr` - `src` 的地址
- `work0_addr` - `work0` 的地址
- `work1_addr` - `work1` 的地址
- `work2_addr` - `work2` 的地址
- `exp_coeff_addr` - `exp_coeff` 的地址
- `erf_coeff_addr` - `erf_coeff` 的地址
- `exp_table_addr` - `exp_table` 的地址
- `shape` - 指向 `dst`、`src`、`work0`、`work1` 和 `work2` 的 `shape` 的指针

注意事项

- `dst`、`src`、`work0`、`work1` 和 `work2` 从同一个 NPU 开始，都是 64-byte aligned layout。

- `exp_coeff` 通过 `tpu_bdc_load_fp32_exp_coeff()` 加载, `erf_coeff` 通过 `tpu_bdc_load_fp32_erf_coeff()` 加载, `exp_table` 通过 `tpu_bdc_load_fp32_exp_table()` 加载。
- `work0`、`work1` 和 `work2` 存放中间结果, 不允许 `dst_addr`、`src_addr`、`work0_addr`、`work1_addr` 和 `work2_addr` 任意两个相等, 如果 `dst`、`work1` 和 `exp_table` 两两之间没有 bank conflicting, 则性能更优。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 $[1, 65535]$, `shape->h * shape->w` 小于等于 65535。
- 如果计算过程中 e^t 的指数 t 小于 -103 或大于 88, 则 e^t 分别是 e^{-103} 和 e^{88} 。

4.18.9 tpu_bdc_fp32_gelu

张量的元素做 GELU 操作。

```
void tpu_bdc_fp32_gelu(local_addr_t dst_addr, local_addr_t src_addr, local_addr_t
    work0_addr, local_addr_t work1_addr, local_addr_t
    work2_addr, local_addr_t work3_addr, local_addr_t
    exp_coeff_addr, local_addr_t erf_coeff_addr, local_addr_t
    exp_table_addr, const dim4 *shape)
```

$$\text{dst}(n, c, h, w) = \frac{1}{2} \left(1 + \text{erf} \left(\frac{\text{src}(n, c, h, w)}{\sqrt{2}} \right) \right)$$

参数

- `dst_addr` - `dst` 的地址
- `src_addr` - `src` 的地址
- `work0_addr` - `work0` 的地址
- `work1_addr` - `work1` 的地址
- `work2_addr` - `work2` 的地址
- `work3_addr` - `work3` 的地址
- `exp_coeff_addr` - `exp_coeff` 的地址
- `erf_coeff_addr` - `erf_coeff` 的地址
- `exp_table_addr` - `exp_table` 的地址
- `shape` - 指向 `dst`、`src`、`work0`、`work1`、`work2` 和 `work3` 的 `shape` 的指针

注意事项

- `dst`、`src`、`work0`、`work1`、`work2` 和 `work3` 从同一个 NPU 开始, 都是 64-byte aligned layout。
- `exp_coeff` 通过 `tpu_bdc_load_fp32_exp_coeff()` 加载, `erf_coeff` 通过 `tpu_bdc_load_fp32_erf_coeff()` 加载, `exp_table` 通过 `tpu_bdc_load_fp32_exp_table()` 加载。
- `work0`、`work1`、`work2` 和 `work3` 存放中间结果, 不允许 `dst_addr`、`work0_addr`、`work1_addr`、`work2_addr` 和 `work3_addr` 任意两个相等, 允许 `src_addr` 等于 `dst_addr`、`work0_addr`、`work1_addr`、`work2_addr` 或 `work3_addr`, 此情况下 `src` 的数据会被覆盖, 如果 `dst`、`work1` 和 `exp_table` 两两之间没有 bank conflicting, 则性能更优。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 $[1, 65535]$, `shape->h * shape->w` 小于等于 65535。
- `erf` 见 `tpu_bdc_fp32_erf()`。

4.18.10 tpu_bdc_fp32_gelu_fast

张量的元素做快速 GELU 操作。

```
void tpu_bdc_fp32_gelu_fast(local_addr_t dst_addr, local_addr_t src_addr,
                           local_addr_t work0_addr, local_addr_t work1_addr,
                           local_addr_t coeff_addr, local_addr_t table_addr, const
                           dim4 *shape)
```

$$\text{dst}(n, c, h, w) = \frac{\text{src}(n, c, h, w)}{2} \times \left(1 + \tanh \left(\sqrt{\frac{2}{\pi}} \times (\text{src}(n, c, h, w) + 0.044715 \times \text{src}(n, c, h, w)^3) \right) \right)$$

参数

- dst_addr -dst 的地址
- src_addr -src 的地址
- work0_addr -work0 的地址
- work1_addr -work1 的地址
- coeff_addr -coeff 的地址
- table_addr -table 的地址
- shape -dst、src、work0 和 work1 的 shape 的指针

注意事项

- dst、src、work0 和 work1 从同一个 NPU 开始，都是 64-byte aligned layout。
- coeff 通过 tpu_bdc_load_fp32_exp_coeff() 加载，table 通过 tpu_bdc_load_fp32_exp_table() 加载。
- work0 和 work1 存放中间结果，不允许 dst_addr、src_addr、work0_addr 和 work1_addr 任意两个相等，如果 dst、work1 和 table 两两之间没有 bank conflict-ing，则性能更优。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]，shape->h * shape->w 小于等于 65535。
- tanh 见 tpu_bdc_fp32_tanh()。

4.18.11 tpu_bdc_fp32_mish

张量的元素做 mish 操作。

```
void tpu_bdc_fp32_mish(local_addr_t dst_addr, local_addr_t src_addr, local_addr_t
                      work0_addr, local_addr_t work1_addr, local_addr_t
                      coeff_addr, local_addr_t table_addr, const dim4 *shape)
```

$$\begin{aligned} \text{dst}(n, c, h, w) &= \text{src}(n, c, h, w) \times \tanh(\log(1 + e^{\text{src}(n, c, h, w)})) \\ &= \text{src}(n, c, h, w) \times \left(1 - 2 \times \frac{1}{(1 + e^{\text{src}(n, c, h, w)})^2 + 1} \right) \end{aligned}$$

参数

- dst_addr -dst 的地址
- src_addr -src 的地址
- work0_addr -work0 的地址
- work1_addr -work1 的地址

- `coeff_addr` -coeff 的地址
- `table_addr` -table 的地址
- `shape` -dst、src、work0 和 work1 的 shape 的指针

注意事项

- dst、src、work0 和 work1 从同一个 NPU 开始，都是 64-byte aligned layout。
- coeff 通过 `tpu_bdc_load_fp32_exp_coeff()` 加载，table 通过 `tpu_bdc_load_fp32_exp_table()` 加载。
- work0 和 work1 存放中间结果，不允许 `dst_addr`、`src_addr`、`work0_addr` 和 `work1_addr` 任意两个相等，如果 dst、work1 和 table 两两之间没有 bank conflict-ing，则性能更优。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]，`shape->h * shape->w` 小于等于 65535。
- 如果 `src(n, c, h, w)` 小于 -103 或大于 88，则 $e^{\text{src}(n, c, h, w)}$ 分别是 e^{-103} 和 e^{88} 。

4.18.12 tpu_bdc_fp32_silu

张量的元素做 SiLU 操作。

```
void tpu_bdc_fp32_silu(local_addr_t dst_addr, local_addr_t src_addr, local_addr_t
                      work0_addr, local_addr_t work1_addr, local_addr_t
                      coeff_addr, local_addr_t table_addr, const dim4 *shape)
```

$$\text{dst}(n, c, h, w) = \text{src}(n, c, h, w) \times \frac{1}{1 + e^{-\text{src}(n, c, h, w)}}$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `work0_addr` -work0 的地址
- `work1_addr` -work1 的地址
- `coeff_addr` -coeff 的地址
- `table_addr` -table 的地址
- `shape` -dst、src、work0 和 work1 的 shape 的指针

注意事项

- dst、src、work0 和 work1 从同一个 NPU 开始，都是 64-byte aligned layout。
- coeff 通过 `tpu_bdc_load_fp32_exp_coeff()` 加载，table 通过 `tpu_bdc_load_fp32_exp_table()` 加载。
- work0 和 work1 存放中间结果，不允许 `dst_addr`、`src_addr`、`work0_addr` 和 `work1_addr` 任意两个相等，如果 dst、work1 和 table 两两之间没有 bank conflict-ing，则性能更优。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]，`shape->h * shape->w` 小于等于 65535。
- 如果 `src(n, c, h, w)` 小于 -88 或大于 103，则 $e^{-\text{src}(n, c, h, w)}$ 分别是 e^{88} 和 e^{-103} 。

4.18.13 tpu_bdc_fp32_selu

张量的元素做 SELU 操作。

```
void tpu_bdc_fp32_selu(local_addr_t dst_addr, local_addr_t src_addr, local_addr_t
                      work0_addr, local_addr_t work1_addr, local_addr_t
                      coeff_addr, local_addr_t table_addr, float alpha, const dim4
                      *shape)
```

$$\text{dst}(n, c, h, w) = \lambda \times \begin{cases} \text{src}(n, c, h, w) & \text{如果 } \text{shift}(n, c, h, w) > 0 \\ \alpha \times (e^{\text{src}(n, c, h, w)} - 1) & \text{其他情况} \end{cases}$$

$$\alpha = 1.6732632423543772848170429916717$$

$$\lambda = 1.0507009873554804934193349852946$$

参数

- dst_addr -dst 的地址
- src_addr -src 的地址
- work0_addr -work0 的地址
- work1_addr -work1 的地址
- coeff_addr -coeff 的地址
- table_addr -table 的地址
- alpha -常数
- shape -dst、src、work0 和 work1 的 shape 的指针

注意事项

- dst、src、work0 和 work1 从同一个 NPU 开始，都是 64-byte aligned layout。
- coeff 通过 tpu_bdc_load_fp32_exp_coeff() 加载，table 通过 tpu_bdc_load_fp32_exp_table() 加载。
- work0 和 work1 存放中间结果，不允许 dst_addr、src_addr、work0_addr 和 work1_addr 任意两个相等，如果 dst、work1 和 table 两两之间没有 bank conflict-ing，则性能更优。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]，shape->h * shape->w 小于等于 65535。
- 如果 src(n, c, h, w) 小于 -103 或大于 88，则 $e^{\text{src}(n, c, h, w)}$ 分别是 e^{-103} 和 e^{88} 。

4.18.14 tpu_bdc_fp_hsigmoid

张量的元素做 HARDSIGMOID 操作。

```
tpu_bdc_fp_hsigmoid(local_addr_t dst_addr, local_addr_t src_addr, local_addr_t
                    work0_addr, const dim4 *shape, float alpha, float beta,
                    data_type_t dtype)
```

```
.. math::
\mathsf{dst}(n, c, h, w) = \mathsf{\lambda \times \begin{cases}
\mathsf{0} \& \mathsf{\text{如果} \sim x \leq -3} \backslash \quad \mathsf{1} \& \mathsf{\text{如果} \sim x \geq 3} \backslash}
\mathsf{\alpha \times \left( \mathsf{src}(n, c, h, w) + \mathsf{\beta} \right)} \& \mathsf{\text{其他情况}} \end{cases}}
.. math:: \mathsf{\alpha = 0.16666666666666666666666666666667}
.. math:: \mathsf{\beta = 0.5}
```

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `work0_addr` -work0 的地址
- `shape` -dst、src、work0 的 shape 的指针
- `alpha` -常数
- `beta` -常数
- `dtype` -dst 和 src 的元素的数据类型

注意事项

- `dst`、`src`、`work0` 从同一个 NPU 开始，都是 64-byte aligned layout。

4.18.15 tpu_bdc_fp_hswish

张量的元素做 HARDSWISH 操作。

```
tpu_bdc_fp_hswish(local_addr_t dst_addr, local_addr_t src_addr, local_addr_t
                  work0_addr, const dim4 *shape, data_type_t dtype)

.. math::
\mathsf{dst}(n, c, h, w) = \mathsf{\lambda \times \begin{cases}
\mathsf{0} \& \mathsf{\text{如果} \sim x \leq -3} \backslash \quad \mathsf{src}(n, c, h, w) \& \mathsf{\text{如果} \sim x \geq 3} \backslash \quad \mathsf{src}(n, c, h, w) \times \left( \mathsf{src}(n, c, h, w) + 0.5 \right) \div 6} \& \mathsf{\text{其他情况}} \end{cases}}
```

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `work0_addr` -work0 的地址
- `shape` -dst、src、work0 的 shape 的指针
- `dtype` -dst 和 src 的元素的数据类型

注意事项

- `dst`、`src`、`work0` 从同一个 NPU 开始，都是 64-byte aligned layout。

4.19 scatter 和 gather 操作**4.19.1 tpu_bdc_w_gather**

通过 `w` 维度的索引取值得到输出张量，即 `output = param[index]`。

```
void tpu_bdc_w_gather(local_addr_t output_addr, local_addr_t param_addr,
                    local_addr_t index_addr, const dim4 *shape, int param_w,
                    data_type_t dtype, data_type_t index_dtype)
```

```
output(n, c, 0, w) = param(n, c, 0, index(0, w, 0, 0))
```

参数

- `output_addr` -output 的地址
- `param_addr` -param 的地址
- `index_addr` -index 的地址
- `shape` -指向 output 的 shape 的指针
- `param_w` -param 的 w
- `dtype` -output 和 param 的元素的数据类型
- `index_dtype` -index 的元素的数据类型

注意事项

- output 和 param 从同一个 NPU 开始，都是 64-byte aligned layout，index 从 NPU 0 开始，compact layout。
- `shape->h` 只能是 1，param 的 shape 是 `[shape->n, shape->c, 1, param_w]`，index 的 shape 是 `[1, shape->w, 1, 1]`。
- `shape->n`、`shape->c`、`shape->w` 和 `param_w` 的取值范围是 `[1, 65535]`。
- `index_dtype` 的有效取值是 `DT_UINT16` 和 `DT_UINT8`，index 的元素的取值范围是 `[0, param_w - 1]`。

4.19.2 tpu_bdc_w_gather_exception

通过 w 维度的索引取值得到输出张量，即 `output = param[index]`，索引的最大值特殊处理。

```
void tpu_bdc_w_gather_exception(local_addr_t output_addr, local_addr_t
                               param_addr, local_addr_t index_addr, scalar_t C,
                               const dim4 *shape, int param_w, data_type_t
                               dtype, data_type_t index_dtype, bool fill_const)
```

$$\text{output}(n, c, 0, w) = \begin{cases} \text{param}(n, c, 0, \text{index}(0, w, 0, 0)) & \text{如果 } \text{index}(0, w, 0, 0) \text{ 不是最大值} \\ C & \text{如果 } \text{index}(0, w, 0, 0) \text{ 是最大值, fill_const 是 true} \end{cases}$$

参数

- `output_addr` -output 的地址
- `param_addr` -param 的地址
- `index_addr` -index 的地址
- `C` -常数
- `shape` -指向 output 的 shape 的指针
- `param_w` -param 的 w
- `dtype` -output 和 param 的元素和 `C` 的数据类型
- `index_dtype` -index 的元素的数据类型
- `fill_const` -output 在索引最大值处填 `C` 的标志

注意事项

- output 和 param 从同一个 NPU 开始，都是 64-byte aligned layout，index 从 NPU 0 开始，compact layout。
- `shape->h` 只能是 1，param 的 shape 是 `[shape->n, shape->c, 1, param_w]`，index 的 shape 是 `[1, shape->w, 1, 1]`。

- `shape->n`、`shape->c`、`shape->w` 和 `param_w` 的取值范围是 $[1, 65535]$ 。
- `index_dtype` 的有效取值是 `DT_UINT16` 和 `DT_UINT8`，`index` 的元素的取值范围是 $[0, \text{param_w} - 1]$ 和索引最大值，如果 `index_dtype` 是 `DT_UINT16`，则索引最大值为 65535，如果 `index_dtype` 是 `DT_UINT8`，则索引最大值为 255。
- 如果索引是最大值，`fill_const` 是 `false`，则 `output` 的对应元素不会被填。

4.19.3 tpu_bdc_w_scatter

通过 `w` 维度的索引改变输出张量的对应元素，即 `output[index] = param`。

```
void tpu_bdc_w_scatter(local_addr_t output_addr, local_addr_t param_addr,
                      local_addr_t index_addr, const dim4 *shape, int param_w,
                      data_type_t dtype, data_type_t index_dtype)
```

$$\text{output}(n, c, 0, \text{index}(0, w, 0, 0)) = \text{param}(n, c, 0, w)$$

参数

- `output_addr` - `output` 的地址
- `param_addr` - `param` 的地址
- `index_addr` - `index` 的地址
- `shape` - 指向 `output` 的 `shape` 的指针
- `param_w` - `param` 的 `w`
- `dtype` - `output` 和 `param` 的元素的数据类型
- `index_dtype` - `index` 的元素的数据类型

注意事项

- `output` 和 `param` 从同一个 NPU 开始，都是 64-byte aligned layout，`index` 从 NPU 0 开始，compact layout。
- `shape->h` 只能是 1，`param` 的 `shape` 是 $[\text{shape->n}, \text{shape->c}, 1, \text{param_w}]$ ，`index` 的 `shape` 是 $[1, \text{param_w}, 1, 1]$ 。
- `shape->n`、`shape->c`、`shape->w` 和 `param_w` 的取值范围是 $[1, 65535]$ 。
- `index_dtype` 的有效取值是 `DT_UINT16` 和 `DT_UINT8`，`index` 的元素的取值范围是 $[0, \text{shape->w} - 1]$ 。

4.19.4 tpu_bdc_hw_gather

通过 `h` 和 `w` 维度的索引取值得到输出张量，即 `output = param[index]`。

```
void tpu_bdc_hw_gather(local_addr_t output_addr, local_addr_t param_addr,
                      local_addr_t index_addr, const dim4 *shape, int param_h, int
                      param_w, data_type_t dtype)
```

$$\text{output}(n, c, h, w) = \text{param}(n, c, h_{\text{param}}, w_{\text{param}})$$

$$h_{\text{param}} = \text{index}(0, h \times W_{\text{output}} + w, 0, 1)$$

$$w_{\text{param}} = \text{index}(0, h \times W_{\text{output}} + w, 0, 0)$$

参数

- `output_addr` - `output` 的地址

- `param_addr` -param 的地址
- `index_addr` -index 的地址
- `shape` -指向 output 的 shape 的指针
- `param_h` -param 的 h
- `param_w` -param 的 w
- `dtype` -output 和 param 的元素的数据类型
- output 和 param 从同一个 NPU 开始, 都是 64-byte aligned layout, index 从 NPU 0 开始, compact layout。
- param 的 shape 是 [shape->n, shape->c, param_h, param_w], index 的 shape 是 [1, shape->h * shape->w, 1, 2]。
- shape->n、shape->c、shape->h、shape->w、param_h 和 param_w 的取值范围是 [1, 65535], shape->h * shape->w 小于等于 65535。
- index 的数据类型是 UINT16, index(0, k, 0, 0) 存放 param 的 w 维度的坐标, 取值范围是 [0, param_w - 1], index(0, k, 0, 1) 存放 param 的 h 维度的坐标, 取值范围是 [0, param_h - 1]。

4.19.5 tpu_bdc_hw_gather_exception

通过 h 和 w 维度的索引取值得到输出张量, 即 `output = param[index]`, 索引的最大值特殊处理。

```
void tpu_bdc_hw_gather_exception(local_addr_t output_addr, local_addr_t
                                param_addr, local_addr_t index_addr, scalar_t C,
                                const dim4 *shape, int param_h, int param_w,
                                data_type_t dtype, bool fill_const)
```

$$\text{output}(n, c, h, w) = \begin{cases} \text{param}(n, c, h_{\text{param}}, w_{\text{param}}) & \text{如果 } h_{\text{param}} \text{ 和 } w_{\text{param}} \text{ 都不是最大值} \\ C & \text{如果 } h_{\text{param}} \text{ 或 } w_{\text{param}} \text{ 是最大值, fill_const 是 true} \end{cases}$$

$$h_{\text{param}} = \text{index}(0, h \times W_{\text{output}} + w, 0, 1)$$

$$w_{\text{param}} = \text{index}(0, h \times W_{\text{output}} + w, 0, 0)$$

参数

- `output_addr` -output 的地址
- `param_addr` -param 的地址
- `index_addr` -index 的地址
- `C` -常数
- `shape` -指向 output 的 shape 的指针
- `param_h` -param 的 h
- `param_w` -param 的 w
- `dtype` -output 和 param 的元素和 `C` 的数据类型
- `fill_const` -output 在索引最大值处填 `C` 的标志
- output 和 param 从同一个 NPU 开始, 都是 64-byte aligned layout, index 从 NPU 0 开始, compact layout。
- param 的 shape 是 [shape->n, shape->c, param_h, param_w], index 的 shape 是 [1, shape->h * shape->w, 1, 2]。

- `shape->n`、`shape->c`、`shape->h`、`shape->w`、`param_h` 和 `param_w` 的取值范围是 $[1, 65535]$ ，`shape->h * shape->w` 小于等于 65535。
- `index` 的数据类型是 `UINT16`，索引最大值为 65535，`index(0, k, 0, 0)` 存放 `param` 的 `w` 维度的坐标，取值范围是 $[0, \text{param_w} - 1]$ 和索引最大值，`index(0, k, 0, 1)` 存放 `param` 的 `h` 维度的坐标，取值范围是 $[0, \text{param_h} - 1]$ 和索引最大值。
- 如果索引是最大值，`fill_const` 是 `false`，则 `output` 的对应元素不会被填。

4.19.6 tpu_bdc_hw_scatter

通过 `h` 和 `w` 维度的索引改变输出张量的某些元素，即 `output[index] = param`。

```
void tpu_bdc_hw_scatter(local_addr_t output_addr, local_addr_t param_addr,
                        local_addr_t index_addr, const dim4 *shape, int param_h, int
                        param_w, data_type_t dtype)
```

$$\text{output}(n, c, h, w) = \text{param}(n, c, h_{\text{param}}, w_{\text{param}})$$

$$h = \text{index}(0, h_{\text{param}} \times W_{\text{param}} + w_{\text{param}}, 0, 1)$$

$$w = \text{index}(0, h_{\text{param}} \times W_{\text{param}} + w_{\text{param}}, 0, 0)$$

参数

- `output_addr` - `output` 的地址
- `param_addr` - `param` 的地址
- `index_addr` - `index` 的地址
- `shape` - 指向 `output` 的 `shape` 的指针
- `param_h` - `param` 的 `h`
- `param_w` - `param` 的 `w`
- `dtype` - `output` 和 `param` 的元素的数据类型
- `output` 和 `param` 从同一个 NPU 开始，都是 64-byte aligned layout，`index` 从 NPU 0 开始，compact layout。
- `param` 的 `shape` 是 $[\text{shape->n}, \text{shape->c}, \text{param_h}, \text{param_w}]$ ，`index` 的 `shape` 是 $[\text{param_h} * \text{param_w}, 1, 2]$ 。
- `shape->n`、`shape->c`、`shape->h`、`shape->w`、`param_h` 和 `param_w` 的取值范围是 $[1, 65535]$ ，`param_h * param_w` 小于等于 65535。
- `index` 的数据类型是 `UINT16`，`index(0, k, 0, 0)` 存放 `output` 的 `w` 维度的坐标，取值范围是 $[0, \text{shape->w} - 1]$ ，`index(0, k, 0, 1)` 存放 `output` 的 `h` 维度的坐标，取值范围是 $[0, \text{shape->h} - 1]$ 。

4.19.7 tpu_bdc_batch_bcast_w_gather

通过 `w` 维度的索引取值得到输出张量，即 `output = param[index]`，`param` 的 `batch` 被广播。

```
void tpu_bdc_batch_bcast_w_gather(local_addr_t output_addr, local_addr_t
                                   param_addr, local_addr_t index_addr, const
                                   dim4 *shape, int param_w, data_type_t dtype,
                                   data_type_t index_dtype, bool
                                   is_param_repeated)
```

$$\text{output}(n, c, 0, w) = \begin{cases} \text{param}(0, c \bmod \text{NPU_NUM}, 0, \text{index}(n, c, 0, w)) & \text{如果 param 重复} \\ \text{param}(0, c, 0, \text{index}(n, c, 0, w)) & \text{其他情况} \end{cases}$$

参数

- output_addr -output 的地址
- param_addr -param 的地址
- index_addr -index 的地址
- shape -指向 output 和 index 的 shape 的指针
- param_w -param 的 w
- dtype -output 和 param 的元素的数据类型
- index_dtype -index 的元素的数据类型
- is_param_repeated -param 重复的标志

注意事项

- output、param 和 index 从同一个 NPU 开始，都是 64-byte aligned layout。
- shape->h 只能是 1，param 的 shape 是 [1, shape->c, 1, param_w]。
- shape->n、shape->c、shape->w 和 param_w 的取值范围是 [1, 65535]。
- index_dtype 的有效取值是 DT_UINT16 和 DT_UINT8，index 的元素的取值范围是 [0, param_w - 1]。
- 如果 output、param 和 index 两两之间没有 bank conflicting，则性能更优，如果 is_param_repeated 是 true，则判断是否冲突时，param 的 size 按照 shape 是 [1, 1, 1, param_w] 计算。

4.19.8 tpu_bdc_batch_bcast_w_gather_exception

通过 w 维度的索引取值得到输出张量，即 $output = param[index]$ ，param 的 batch 被广播，索引的最大值特殊处理。

```
void tpu_bdc_batch_bcast_w_gather_exception(local_addr_t output_addr,
                                             local_addr_t param_addr,
                                             local_addr_t index_addr, scalar_t C,
                                             const dim4 *shape, int param_w,
                                             data_type_t dtype, data_type_t
                                             index_dtype, bool is_param_repeated,
                                             bool fill_const)
```

$$output(n, c, 0, w) = \begin{cases} param(0, s, 0, index(n, c, 0, w)) & \text{如果 } index(n, c, 0, w) \text{ 不是最大值} \\ C & \text{如果 } index(n, c, 0, w) \text{ 是最大值, } fill_const \text{ 是 true} \end{cases}$$

$$s = \begin{cases} c \bmod NPU_NUM & \text{如果 param 重复} \\ c & \text{其他情况} \end{cases}$$

参数

- output_addr -output 的地址
- param_addr -param 的地址
- index_addr -index 的地址
- C -常数
- shape -指向 output 和 index 的 shape 的指针
- param_w -param 的 w
- dtype -output 和 param 的元素和 C 的数据类型

- `index_dtype` -index 的元素的数据类型
- `is_param_repeated` -param 重复的标志
- `fill_const` -output 在索引最大值处填 C 的标志

注意事项

- output、param 和 index 从同一个 NPU 开始，都是 64-byte aligned layout。
- `shape->h` 只能是 1，param 的 shape 是 `[1, shape->c, 1, param_w]`。
- `shape->n`、`shape->c`、`shape->w` 和 `param_w` 的取值范围是 `[1, 65535]`。
- `index_dtype` 的有效取值是 `DT_UINT16` 和 `DT_UINT8`，index 的元素的取值范围是 `[0, param_w - 1]` 和索引最大值，如果 `index_dtype` 是 `DT_UINT16`，则索引最大值为 65535，如果 `index_dtype` 是 `DT_UINT8`，则索引最大值为 255。
- 如果索引是最大值，`fill_const` 是 false，则 output 的对应元素不会被填。
- 如果 output、param 和 index 两两之间没有 bank conflicting，则性能更优，如果 `is_param_repeated` 是 true，则判断是否冲突时，param 的 size 按照 shape 是 `[1, 1, 1, param_w]` 计算。

4.19.9 tpu_bdc_batch_bcast_w_scatter

通过 w 维度的索引改变输出张量的对应元素，即 `output[index] = param`，param 的 batch 被广播。

```
void tpu_bdc_batch_bcast_w_scatter(local_addr_t output_addr, local_addr_t
                                   param_addr, local_addr_t index_addr, const
                                   dim4 *shape, int param_w, data_type_t dtype,
                                   data_type_t index_dtype, bool
                                   is_param_repeated)
```

$$\text{output}(n, c, 0, \text{index}(n, c, 0, w)) = \begin{cases} \text{param}(0, c \bmod \text{NPU_NUM}, 0, w) & \text{如果 param 重复} \\ \text{param}(0, c, 0, w) & \text{其他情况} \end{cases}$$

参数

- `output_addr` -output 的地址
- `param_addr` -param 的地址
- `index_addr` -index 的地址
- `shape` -指向 output 的 shape 的指针
- `param_w` -param 的 w
- `dtype` -output 和 param 的元素的数据类型
- `index_dtype` -index 的元素的数据类型
- `is_param_repeated` -param 重复的标志

注意事项

- output、param 和 index 从同一个 NPU 开始，都是 64-byte aligned layout。
- `shape->h` 只能是 1，param 的 shape 是 `[1, shape->c, 1, param_w]`，index 的 shape 是 `[shape->n, shape->c, 1, param_w]`。
- `shape->n`、`shape->c`、`shape->w` 和 `param_w` 的取值范围是 `[1, 65535]`。
- `index_dtype` 的有效取值是 `DT_UINT16` 和 `DT_UINT8`，index 的元素的取值范围是 `[0, shape->w - 1]`。

- 如果 output、param 和 index 两两之间没有 bank conflicting, 则性能更优, 如果 is_param_repeated 是 true, 则判断是否冲突时, param 的 size 按照 shape 是 [1, 1, 1, param_w] 计算。

4.19.10 tpu_bdc_batch_bcast_w_mask_select

通过 w 维度的蒙版取值得到输出张量, 即 $\text{output} = \text{param}[\text{mask}]$, 并统计蒙版中的非零值数量, param 的 batch 被广播。

```
void tpu_bdc_batch_bcast_w_mask_select(local_addr_t output_addr, local_addr_t
                                       count_addr, local_addr_t param_addr,
                                       local_addr_t mask_addr, const dim4
                                       *shape, data_type_t dtype, data_type_t
                                       mask_dtype, bool is_param_repeated)
```

对于固定的 n 和 c 有 R 个非零值: $\text{mask}(n, c, 0, w_0), \text{mask}(n, c, 0, w_1), \dots, \text{mask}(n, c, 0, w_{R-1})$

$$\text{output}(n, c, 0, r) = \begin{cases} \text{param}(0, c \bmod \text{NPU_NUM}, 0, w_r) & \text{如果 param 重复} \\ \text{param}(0, c, 0, w_r) & \text{其他情况} \end{cases}$$

$$\text{count}(n, c, 0, 0) = R$$

参数

- output_addr - output 的地址
- count_addr - count 的地址
- param_addr - param 的地址
- mask_addr - mask 的地址
- shape - 指向 output 和 mask 的 shape 的指针
- dtype - output 和 param 的元素的数据类型
- mask_dtype - mask 的元素的数据类型
- is_param_repeated - param 重复的标志

注意事项

- output、count、param 和 mask 从同一个 NPU 开始, output、param 和 mask 是 64-byte aligned layout, count 是 compact layout。
- shape->h 只能是 1, count 的 shape 是 [shape->n, shape->c, 1, 1], param 的 shape 是 [1, shape->c, 1, shape->w]。
- shape->n、shape->c、shape->w 的取值范围是 [1, 65535]。
- mask_dtype 的有效取值是 DT_UINT32、DT_UINT16 和 DT_UINT8, count 的元素的数据类型是 UINT16, 输出范围是 [0, shape->w]。
- 如果 output、count、param 和 mask 两两之间没有 bank conflicting, 则性能更优, 如果 is_param_repeated 是 true, 则判断是否冲突时, param 的 size 按照 shape 是 [1, 1, 1, shape->w] 计算。

4.19.11 tpu_bdc_batch_bcast_h_gather

通过 h 维度的索引取值得到输出张量，即 $output = param[index]$ ，param 的 batch 被广播。

```
void tpu_bdc_batch_bcast_h_gather(local_addr_t output_addr, local_addr_t
                                param_addr, local_addr_t index_addr, const
                                dim4 *shape, int param_h, data_type_t dtype,
                                data_type_t index_dtype, bool
                                is_param_repeated)
```

$$output(n, c, h, w) = \begin{cases} param(0, c \bmod NPU_NUM, index(n, c, h, 0), w) & \text{如果 param 重复} \\ param(0, c, index(n, c, h, 0), w) & \text{其他情况} \end{cases}$$

参数

- output_addr - output 的地址
- param_addr - param 的地址
- index_addr - index 的地址
- shape - 指向 output 的 shape 的指针
- param_h - param 的 h
- dtype - output 和 param 的元素的数据类型
- index_dtype - index 的元素的数据类型
- is_param_repeated - param 重复的标志

注意事项

- output、param 和 index 从同一个 NPU 开始，output 和 param 是 line 64-byte aligned layout，index 是 64-byte aligned layout。
- param 的 shape 是 $[1, shape->c, param_h, shape->w]$ ，index 的 shape 是 $[shape->n, shape->c, shape->h, 1]$ 。
- shape->n、shape->c、shape->h、shape->w 和 param_h 的取值范围是 $[1, 65535]$ 。
- index_dtype 的有效取值是 DT_UINT16 和 DT_UINT8，index 的元素的取值范围是 $[0, param_h - 1]$ 。

4.19.12 tpu_bdc_batch_bcast_h_gather_exception

通过 h 维度的索引取值得到输出张量，即 $output = param[index]$ ，param 的 batch 被广播，索引的最大值特殊处理。

```
void tpu_bdc_batch_bcast_h_gather_exception(local_addr_t output_addr,
                                             local_addr_t param_addr,
                                             local_addr_t index_addr, scalar_t C,
                                             const dim4 *shape, int param_h,
                                             data_type_t dtype, data_type_t
                                             index_dtype, bool is_param_repeated,
                                             bool fill_const)
```

$$output(n, c, h, w) = \begin{cases} param(0, s, index(n, c, h, 0), w) & \text{如果 index}(n, c, h, 0) \text{ 不是最大值} \\ C & \text{如果 index}(n, c, h, 0) \text{ 是最大值, fill_const 是 true} \end{cases}$$

$$s = \begin{cases} c \bmod NPU_NUM & \text{如果 param 重复} \\ c & \text{其他情况} \end{cases}$$

参数

- `output_addr` -output 的地址
- `param_addr` -param 的地址
- `index_addr` -index 的地址
- `C` -常数
- `shape` -指向 output 的 shape 的指针
- `param_h` -param 的 h
- `dtype` -output 和 param 的元素和 `C` 的数据类型
- `index_dtype` -index 的元素的数据类型
- `is_param_repeated` -param 重复的标志
- `fill_const` -output 在索引最大值处填 `C` 的标志

注意事项

- output、param 和 index 从同一个 NPU 开始，output 和 param 是 line 64-byte aligned layout，index 是 64-byte aligned layout。
- param 的 shape 是 [1, shape->c, param_h, shape->w]，index 的 shape 是 [shape->n, shape->c, shape->h, 1]。
- shape->n、shape->c、shape->h、shape->w 和 param_h 的取值范围是 [1, 65535]。
- index_dtype 的有效取值是 `DT_UINT16` 和 `DT_UINT8`，index 的元素的取值范围是 [0, param_h - 1] 和索引最大值，如果 index_dtype 是 `DT_UINT16`，则索引最大值为 65535，如果 index_dtype 是 `DT_UINT8`，则索引最大值为 255。
- 如果索引是最大值，fill_const 是 false，则 output 的对应元素不会被填。

4.19.13 tpu_bdc_batch_bcast_h_scatter

通过 h 维度的索引改变输出张量的对应元素，即 `output[index] = param`，param 的 batch 被广播。

```
void tpu_bdc_batch_bcast_h_scatter(local_addr_t output_addr, local_addr_t
                                   param_addr, local_addr_t index_addr, const
                                   dim4 *shape, int param_h, data_type_t dtype,
                                   data_type_t index_dtype, bool
                                   is_param_repeated)
```

$$\text{output}(n, c, \text{index}(n, c, h, 0), w) = \begin{cases} \text{param}(0, c \bmod \text{NPU_NUM}, h, w) & \text{如果 param 重复} \\ \text{param}(0, c, h, w) & \text{其他情况} \end{cases}$$

参数

- `output_addr` -output 的地址
- `param_addr` -param 的地址
- `index_addr` -index 的地址
- `shape` -指向 output 的 shape 的指针
- `param_h` -param 的 h
- `dtype` -output 和 param 的元素的数据类型
- `index_dtype` -index 的元素的数据类型
- `is_param_repeated` -param 重复的标志

注意事项

- output、param 和 index 从同一个 NPU 开始，output 和 param 是 line 64-byte aligned layout，index 是 64-byte aligned layout。
- param 的 shape 是 [1, shape->c, param_h, shape->w]，index 的 shape 是 [shape->n, shape->c, param_h, 1]。
- shape->n、shape->c、shape->h、shape->w 和 param_h 的取值范围是 [1, 65535]。
- index_dtype 的有效取值是 DT_UINT16 和 DT_UINT8，index 的元素的取值范围是 [0, param_h - 1]。

4.20 特殊函数

4.20.1 tpu_bdc_fp_taylor

张量的元素的 Taylor 级数。

```
void tpu_bdc_fp_taylor(local_addr_t dst_addr, local_addr_t src_addr, local_addr_t
                      coeff_addr, const dim4 *shape, int num, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \sum_{i=0}^{\text{num}-1} \text{coeff}(0, c \bmod \text{NPU_NUM}, 0, i) \times \text{src}(n, c, h, w)^i$$

参数

- dst_addr -dst 的地址
- src_addr -src 的地址
- coeff_addr -coeff 的地址
- shape -指向 dst 和 src 的 shape 的指针
- num -Taylor 级数的项数
- dtype -dst、src 和 coeff 的元素的数据类型

注意事项

- dst 和 src 从同一个 NPU 开始，都是 64-byte aligned layout。
- coeff 从 NPU 0 开始，shape 是 [1, NPU_NUM, 1, num]，64-byte aligned layout。
- num 大于等于 2。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。

4.20.2 tpu_bdc_table_lookup

通过张量的元素查表。

```
void tpu_bdc_table_lookup(local_addr_t dst_addr, local_addr_t src_addr,
                          local_addr_t table_addr, const dim4 *shape, int len,
                          data_type_t dst_dtype, data_type_t src_dtype)
```

$$\text{dst}(n, c, h, w) = \text{table}(0, c \bmod \text{NPU_NUM}, 0, \text{src}(n, c, h, w))$$

参数

- dst_addr -dst 的地址

- `src_addr` -src 的地址
- `table_addr` -table 的地址
- `shape` -指向 dst 和 src 的 shape 的指针
- `len` -表的长度
- `dst_dtype` -dst 和 table 的元素的数据类型
- `src_dtype` -src 的元素的数据类型

注意事项

- dst 和 src 从同一个 NPU 开始，都是 64-byte aligned layout。
- table 从 NPU 0 开始，shape 是 [1, `NPU_NUM`, 1, `len`]，64-byte aligned layout。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]，`shape->h * shape->w` 小于等于 65535。
- `src_dtype` 的有效取值是 `DT_UINT16` 和 `DT_UINT8`，src 的元素的取值范围是 [0, `len` - 1]。
- 如果 dst、src 和 table 两两之间没有 bank conflicting，则性能更优。

4.20.3 tpu_bdc_arithmetic_sequence_bcast

生成等差数列，广播到各个 NPU。

```
void tpu_bdc_arithmetic_sequence_bcast(local_addr_t dst_addr, int npu_num, int
                                     start, int step, int num)
```

$$\text{dst}(0, c, 0, w) = \text{start} + \text{step} \times w$$

参数

- `dst_addr` -dst 的地址
- `npu_num` -广播 NPU 的数量
- `start` -等差数列的首项
- `step` -等差数列的步长
- `num` -等差数列的项数

注意事项

- dst 的 shape 是 [1, `npu_num`, 1, `num`]，64-byte aligned layout，元素的数据类型是 INT32。
- 如果 dst 从 NPU X 开始，X 的取值范围是 [0, `NPU_NUM` - 1]，则 `npu_num` 小于等于 `NPU_NUM` - X。
- `num` 大于 0。

4.20.4 tpu_bdc_arithmetic_sequence_distribute

生成等差数列，分布在各个 NPU。

```
void tpu_bdc_arithmetic_sequence_distribute(local_addr_t dst_addr, int start, int step,
                                             int num)
```

$$\text{dst}(0, c, 0, 0) = \text{start} + \text{step} \times c$$

参数

- `dst_addr` -dst 的地址
- `start` -等差数列的首项
- `step` -等差数列的步长
- `num` -等差数列的项数

注意事项

- `dst` 从 NPU 0 开始，compact layout。
- `dst` 的 shape 是 `[1, num, 1, 1]`，元素的数据类型是 INT32。
- `num` 大于 0。

4.20.5 tpu_bdc_arithmetic_sequence_general

生成等差数列，分布在各个 NPU。

```
void tpu_bdc_arithmetic_sequence_general(local_addr_t dst_addr, local_addr_t
                                          buffer_addr, int npu_num, int start, int
                                          step, int num)
```

$$\text{dst}(0, c, 0, w) = \text{start} + \text{step} \times (\text{num} \times c + w)$$

参数

- `dst_addr` -dst 的地址
- `buffer_addr` -buffer 的地址
- `npu_num` -序列数 channel 的数量
- `start` -等差数列的首项
- `step` -等差数列的步长
- `num` -等差数列的项数

注意事项

- `dst` 的 shape 是 `[1, npu_num, 1, num]`，64-byte aligned layout，元素的数据类型是 INT32。
- 要求 `dst` 和 `buffer` 从 NPU 0 开始。
- `buffer` 的 shape 是 `[1, NPU_NUM, 1, 1]`，compact layout，元素的数据类型是 INT32。
- `npu_num` 大于 0。
- `num` 大于 0。

4.20.6 tpu_bdc_load_fp32_exp_coeff

加载 exp 的 Taylor 系数到 local memory。

```
void tpu_bdc_load_fp32_exp_coeff(local_addr_t coeff_addr)
```

参数 coeff_addr -coeff 的地址

注意事项

- coeff 从 NPU 0 开始, shape 是 [1, NPU_NUM, 1, 32], 64-byte aligned layout。

4.20.7 tpu_bdc_load_fp32_exp_table

加载 -103 ~ 88 对应的 exp 的表到 local memory。

```
void tpu_bdc_load_fp32_exp_table(local_addr_t table_addr)
```

参数 table_addr -table 的地址

注意事项

- table 从 NPU 0 开始, shape 是 [1, NPU_NUM, 1, 192], 64-byte aligned layout。

4.20.8 tpu_bdc_load_fp32_log_coeff

加载 log 的 Taylor 系数到 local memory。

```
void tpu_bdc_load_fp32_log_coeff(local_addr_t coeff_addr)
```

参数 coeff_addr -coeff 的地址

注意事项

- coeff 从 NPU 0 开始, shape 是 [1, NPU_NUM, 1, 32], 64-byte aligned layout。

4.20.9 tpu_bdc_load_fp32_erf_coeff

加载 erf 的 Taylor 系数到 local memory。

```
void tpu_bdc_load_fp32_erf_coeff(local_addr_t coeff_addr)
```

参数 coeff_addr -coeff 的地址

注意事项

- coeff 从 NPU 0 开始, shape 是 [1, NPU_NUM, 1, 10], 64-byte aligned layout。

4.20.10 tpu_bdc_load_fp32_sin_coeff

加载 sin 的 Taylor 系数到 local memory。

```
void tpu_bdc_load_fp32_sin_coeff(local_addr_t coeff_addr)
```

参数 `coeff_addr` -coeff 的地址

注意事项

- `coeff` 从 NPU 0 开始, shape 是 `[1, NPU_NUM, 1, 32]`, 64-byte aligned layout。

4.20.11 tpu_bdc_load_fp32_cos_coeff

加载 cos 的 Taylor 系数到 local memory。

```
void tpu_bdc_load_fp32_cos_coeff(local_addr_t coeff_addr)
```

参数 `coeff_addr` -coeff 的地址

注意事项

- `coeff` 从 NPU 0 开始, shape 是 `[1, NPU_NUM, 1, 32]`, 64-byte aligned layout。

4.20.12 tpu_bdc_load_fp32_tan_coeff

加载 tan 的 Taylor 系数到 local memory。

```
void tpu_bdc_load_fp32_tan_coeff(local_addr_t coeff_addr)
```

参数 `coeff_addr` -coeff 的地址

注意事项

- `coeff` 从 NPU 0 开始, shape 是 `[1, NPU_NUM, 1, 32]`, 64-byte aligned layout。

4.20.13 tpu_bdc_load_fp32_arcsin_coeff

加载 arcsin 的 Taylor 系数到 local memory。

```
void tpu_bdc_load_fp32_arcsin_coeff(local_addr_t coeff_addr)
```

参数 `coeff_addr` -coeff 的地址

注意事项

- `coeff` 从 NPU 0 开始, shape 是 `[1, NPU_NUM, 1, 32]`, 64-byte aligned layout。

4.21 量化操作

4.21.1 tpu_bdc_int_requant

重量化张量的元素，结果有 saturation。

```
void tpu_bdc_int_requant(local_addr_t dst_addr, local_addr_t src_addr, const dim4
                        *shape, int multiplier, char shift, scalar_t offset, data_type_t
                        dst_dtype, data_type_t src_dtype, rounding_mode_t
                        rounding_mode)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (\text{src}(n, c, h, w) \times \text{multiplier} \text{ 左移 } \text{shift}) + \text{offset} & \text{如果 } \text{shift} > 0 \\ (\text{src}(n, c, h, w) \times \text{multiplier} \text{ 右移 } -\text{shift}) + \text{offset} & \text{其他情况} \end{cases}$$

参数

- dst_addr -dst 的地址
- src_addr -src 的地址
- shape -指向 dst 和 src 的 shape 的指针
- multiplier -乘子常数
- shift -移位数
- offset -补偿常数
- dst_dtype -dst 的元素的数据类型
- src_dtype -src 的元素的数据类型
- rounding_mode -右移舍入模式

注意事项

- dst 和 src 从同一个 NPU 开始，都是 64-byte aligned layout。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- dst_dtype 的有效取值是DT_INT16、DT_UINT16、DT_INT8 和DT_UINT8，src_dtype 的有效取值是DT_INT32、DT_INT16 和DT_UINT16，如果 dst_dtype 是有符号的，则 offset 的数据类型是DT_INT16，否则是DT_UINT16。
- shift 的取值范围是 [-64, 31]。

4.21.2 tpu_bdc_int_pc_requant

按 channel 重量化张量的元素，结果有 saturation。

```
void tpu_bdc_int_pc_requant(local_addr_t dst_addr, local_addr_t src_addr,
                           local_addr_t quant_addr, const dim4 *shape,
                           data_type_t dst_dtype, data_type_t src_dtype,
                           rounding_mode_t rounding_mode)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (\text{src}(n, c, h, w) \times \text{quant}(0, c, 0, 0) \text{ 左移 } \text{quant}(0, c, 0, 1)) + \text{quant}(0, c, 0, 2) & \text{如果 } \text{quant}(0, c, 0, 1) > 0 \\ (\text{src}(n, c, h, w) \times \text{quant}(0, c, 0, 0) \text{ 右移 } -\text{quant}(0, c, 0, 1)) + \text{quant}(0, c, 0, 2) & \text{其他情况} \end{cases}$$

参数

- dst_addr -dst 的地址
- src_addr -src 的地址

- `quant_addr` - quant 的地址
- `shape` - 指向 dst 和 src 的 shape 的指针
- `dst_dtype` - dst 的元素的数据类型
- `src_dtype` - src 的元素的数据类型
- `rounding_mode` - 右移舍入模式

注意事项

- dst、src 和 quant 从同一个 NPU 开始，都是 64-byte aligned layout。
- quant 的 shape 是 [1, shape->c, 1, 3]，元素的数据类型是 `DT_INT32`，quant(0, c, 0, 0) 是乘子，quant(0, c, 0, 1) 是移位数，取值范围是 [-64, 31]，quant(0, c, 0, 2) 是补偿，如果 `dst_dtype` 是有符号的，则 quant(0, c, 0, 2) 的取值范围是 [-32768, 32767]，否则是 [0, 65535]。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- `dst_dtype` 的有效取值是 `DT_INT16`、`DT_UINT16`、`DT_INT8` 和 `DT_UINT8`，`src_dtype` 的有效取值是 `DT_INT32`、`DT_INT16` 和 `DT_UINT16`。

4.21.3 tpu_bdc_fp32_requant

重量化张量的元素，结果有 saturation。

```
void tpu_bdc_fp32_requant(local_addr_t dst_addr, local_addr_t src_addr, const dim4
                          *shape, float scale, float offset, data_type_t dst_dtype,
                          data_type_t src_dtype, rounding_mode_t
                          src_rounding_mode, rounding_mode_t
                          dst_rounding_mode)
```

$$\text{dst}(n, c, h, w) = \text{INT}(\text{FP32}(\text{src}(n, c, h, w)) \times \text{scale} + \text{offset})$$

参数

- `dst_addr` - dst 的地址
- `src_addr` - src 的地址
- `shape` - 指向 dst 和 src 的 shape 的指针
- `scale` - 乘子常数
- `offset` - 补偿常数
- `dst_dtype` - dst 的元素的数据类型
- `src_dtype` - src 的元素的数据类型
- `dst_rounding_mode` - 浮点数转化到 dst 的元素的舍入模式
- `src_rounding_mode` - src 的元素转化到浮点数的舍入模式

注意事项

- dst 和 src 从同一个 NPU 开始，都是 64-byte aligned layout。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- `dst_dtype` 的有效取值是 `DT_INT16`、`DT_UINT16`、`DT_INT8` 和 `DT_UINT8`，`src_dtype` 的有效取值是 `DT_INT32`、`DT_INT16` 和 `DT_UINT16`。
- `dst_rounding_mode` 的有效取值是 `RM_HALF_TO_EVEN`、`RM_HALF_AWAY_FROM_ZERO`、`RM_TOWARDS_ZERO`、`RM_DOWN` 和 `RM_UP`。

4.21.4 tpu_bdc_fp32_pc_requant

按 channel 重量化张量的元素，结果有 saturation。

```
void tpu_bdc_fp32_pc_requant(local_addr_t dst_addr, local_addr_t src_addr,
                             local_addr_t quant_addr, const dim4 *shape,
                             data_type_t dst_dtype, data_type_t src_dtype,
                             rounding_mode_t dst_rounding_mode,
                             rounding_mode_t src_rounding_mode)
```

$$\text{dst}(n, c, h, w) = \text{INT}(\text{FP32}(\text{src}(n, c, h, w)) \times \text{quant}(0, c, 0, 0) + \text{quant}(0, c, 0, 1))$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `quant_addr` -quant 的地址
- `shape` -指向 dst 和 src 的 shape 的指针
- `dst_dtype` -dst 的元素的数据类型
- `src_dtype` -src 的元素的数据类型
- `dst_rounding_mode` -浮点数转化到 dst 的元素的舍入模式
- `src_rounding_mode` -src 的元素转化到浮点数的舍入模式

注意事项

- dst、src 和 quant 从同一个 NPU 开始，都是 64-byte aligned layout。
- quant 的 shape 是 [1, shape->c, 1, 2]，元素的数据类型是 FP32，quant(0, c, 0, 0) 是乘子，quant(0, c, 0, 1) 是补偿。
- shape->n、shape->c、shape->h 和 shape->w 的取值范围是 [1, 65535]。
- `dst_dtype` 的有效取值是 `DT_INT16`、`DT_UINT16`、`DT_INT8` 和 `DT_UINT8`，`src_dtype` 的有效取值是 `DT_INT32`、`DT_INT16` 和 `DT_UINT16`。
- `dst_rounding_mode` 的有效取值是 `RM_HALF_TO_EVEN`、`RM_HALF_AWAY_FROM_ZERO`、`RM_TOWARDS_ZERO`、`RM_DOWN` 和 `RM_UP`。

4.21.5 tpu_bdc_int_dequant

反量化张量的元素，结果有 saturation。

```
void tpu_bdc_int_dequant(local_addr_t dst_addr, local_addr_t src_addr, const dim4
                        *shape, scalar_t offset, int multiplier, char shift, data_type_t
                        dst_dtype, data_type_t src_dtype, rounding_mode_t
                        rounding_mode)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (\text{src}(n, c, h, w) - \text{offset}) \times \text{multiplier} \text{ 左移 } \text{shift} & \text{如果 } \text{shift} > 0 \\ (\text{src}(n, c, h, w) - \text{offset}) \times \text{multiplier} \text{ 右移 } -\text{shift} & \text{其他情况} \end{cases}$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `shape` -指向 dst 和 src 的 shape 的指针

- `offset` –补偿常数
- `multiplier` –乘子常数
- `shift` –移位数
- `dst_dtype` –dst 的元素的数据类型
- `src_dtype` –src 的元素的数据类型
- `rounding_mode` –右移舍入模式

注意事项

- `dst` 和 `src` 从同一个 NPU 开始，都是 64-byte aligned layout。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。
- `dst_dtype` 的有效取值是 `DT_INT32`、`DT_INT16` 和 `DT_UINT16`，`src_dtype` 的有效取值是 `DT_INT16`、`DT_UINT16`、`DT_INT8` 和 `DT_UINT8`，如果 `src_dtype` 是有符号的，则 `offset` 的数据类型是 `DT_INT16`，否则是 `DT_UINT16`。
- `shift` 的取值范围是 [-64, 31]。

4.21.6 tpu_bdc_int_pc_dequant

按 channel 反量化张量的元素，结果有 saturation。

```
void tpu_bdc_int_pc_dequant(local_addr_t dst_addr, local_addr_t src_addr,
                           local_addr_t quant_addr, const dim4 *shape,
                           data_type_t dst_dtype, data_type_t src_dtype,
                           rounding_mode_t rounding_mode)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (\text{src}(n, c, h, w) - \text{quant}(0, c, 0, 0)) \times \text{quant}(0, c, 0, 1) \text{ 左移 } \text{quant}(0, c, 0, 2) \\ \text{如果 } \text{quant}(0, c, 0, 2) > 0 \\ (\text{src}(n, c, h, w) - \text{quant}(0, c, 0, 0)) \times \text{quant}(0, c, 0, 1) \text{ 右移 } -\text{quant}(0, c, 0, 2) \text{ 其他情况} \end{cases}$$

参数

- `dst_addr` –dst 的地址
- `src_addr` –src 的地址
- `quant_addr` –quant 的地址
- `shape` –指向 `dst` 和 `src` 的 `shape` 的指针
- `dst_dtype` –dst 的元素的数据类型
- `src_dtype` –src 的元素的数据类型
- `rounding_mode` –右移舍入模式

注意事项

- `dst`、`src` 和 `quant` 从同一个 NPU 开始，都是 64-byte aligned layout。
- `quant` 的 `shape` 是 [1, `shape->c`, 1, 3]，元素的数据类型是 `DT_INT32`，`quant(0, c, 0, 0)` 是补偿，如果 `src_dtype` 是有符号的，则 `quant(0, c, 0, 0)` 的取值范围是 [-32768, 32767]，否则是 [0, 65535]，`quant(0, c, 0, 1)` 是乘子，`quant(0, c, 0, 2)` 是移位数，取值范围是 [-64, 31]。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。
- `dst_dtype` 的有效取值是 `DT_INT32`、`DT_INT16` 和 `DT_UINT16`，`src_dtype` 的有效取值是 `DT_INT16`、`DT_UINT16`、`DT_INT8` 和 `DT_UINT8`。

4.21.7 tpu_bdc_fp32_dequant

反量化张量的元素。

```
void tpu_bdc_fp32_dequant(local_addr_t dst_addr, local_addr_t src_addr, const dim4
                          *shape, scalar_t offset, float scale, data_type_t src_dtype,
                          rounding_mode_t rounding_mode)
```

$$\text{dst}(n, c, h, w) = \text{FP32}(\text{src}(n, c, h, w) - \text{offset}) \times \text{scale}$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `shape` -指向 dst 和 src 的 shape 的指针
- `offset` -补偿常数
- `scale` -乘子常数
- `src_dtype` -src 的元素和 `offset` 的数据类型
- `rounding_mode` -定点数转化到浮点数的舍入模式

注意事项

- dst 和 src 从同一个 NPU 开始，都是 64-byte aligned layout。
- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 [1, 65535]。
- `src_dtype` 的有效取值是 `DT_INT16`、`DT_UINT16`、`DT_INT8` 和 `DT_UINT8`。

4.21.8 tpu_bdc_fp32_pc_dequant

按 channel 反量化张量的元素。

```
void tpu_bdc_fp32_pc_dequant(local_addr_t dst_addr, local_addr_t src_addr,
                             local_addr_t quant_addr, const dim4 *shape,
                             data_type_t src_dtype, rounding_mode_t
                             rounding_mode)
```

$$\text{dst}(n, c, h, w) = \text{FP32}(\text{src}(n, c, h, w) - \text{quant}(0, c, 0, 0)) \times \text{quant}(0, c, 0, 1)$$

参数

- `dst_addr` -dst 的地址
- `src_addr` -src 的地址
- `quant_addr` -quant 的地址
- `shape` -指向 dst 和 src 的 shape 的指针
- `src_dtype` -src 的元素的数据类型
- `rounding_mode` -定点数转化到浮点数的舍入模式

注意事项

- dst、src 和 quant 从同一个 NPU 开始，都是 64-byte aligned layout。
- quant 的 shape 是 [1, `shape->c`, 1, 2]，元素的数据类型是 `DT_INT32` / `DT_FP32`，`quant(0, c, 0, 0)` 是补偿，数据类型是 `DT_INT32`，取值范围与 `src_dtype` 的取值范围相同，`quant(0, c, 0, 1)` 是乘子，数据类型是 `FP32`。

- `shape->n`、`shape->c`、`shape->h` 和 `shape->w` 的取值范围是 $[1, 65535]$ 。`src_dtype` 的有效取值是 `DT_INT16`、`DT_UINT16`、`DT_INT8` 和 `DT_UINT8`。

4.22 HAU 操作

4.22.1 tpu_hau_sort

按升序或降序排序前 K 个最小或最大的数。

```
tpu_hau_sort(system_addr_t output_addr, system_addr_t input_addr, int len, int K,
             bool descended, data_type_t dtype)
```

$$\text{output}(k) = \text{input}(i_k)$$

如果升序，则

$$\text{input}(i_0) \leq \text{input}(i_1) \leq \cdots \leq \text{input}(i_{K-1}) \leq \cdots \leq \text{input}(i_{\text{len}-1})$$

如果降序，则

$$\text{input}(i_0) \geq \text{input}(i_1) \geq \cdots \geq \text{input}(i_{K-1}) \geq \cdots \geq \text{input}(i_{\text{len}-1})$$

其中， $i_0, i_1, \dots, i_{\text{len}-1}$ 互不相同，是 $0, 1, \dots, \text{len} - 1$ 的重排。

参数

- `output_addr` - output 的地址
- `input_addr` - input 的地址
- `len` - input 的长度
- `K` - 排序长度
- `descended` - 降序的标志
- `dtype` - output 和 input 的元素的数据类型

注意事项

- `output_addr` 和 `input_addr` 都被 4 整除。
- `dtype` 的有效取值是 `DT_FP32`、`DT_INT32` 和 `DT_UINT32`。
- output 的长度是 `len`，前 K 个数是排序后的结果， K 小于等于 `len`。

4.22.2 tpu_hau_sort_natural_index

按升序或降序稳定排序前 K 个最小或最大的数，并输出排序后的索引，排序前的索引是自然索引。

```
tpu_hau_sort_natural_index(system_addr_t output_data_addr, system_addr_t
                           output_idx_addr, system_addr_t input_addr, int len, int
                           K, bool descended, data_type_t dtype)
```

$$\text{output_data}(k) = \text{input}(i_k) \quad \text{output_idx}(k) = i_k$$

如果 $\text{input}(i_k) = \text{input}(i_{k+1})$, 则 $i_k < i_{k+1}$

如果升序, 则

$$\text{input}(i_0) \leq \text{input}(i_1) \leq \cdots \leq \text{input}(i_{K-1}) \leq \cdots \leq \text{input}(i_{\text{len}-1})$$

如果降序, 则

$$\text{input}(i_0) \geq \text{input}(i_1) \geq \cdots \geq \text{input}(i_{K-1}) \geq \cdots \geq \text{input}(i_{\text{len}-1})$$

其中, $i_0, i_1, \dots, i_{\text{len}-1}$ 互不相同, 是 $0, 1, \dots, \text{len} - 1$ 的重排。

参数

- `output_data_addr` - `output_data` 的地址
- `output_idx_addr` - `output_idx` 的地址
- `input_addr` - `input` 的地址
- `len` - `input` 的长度
- `K` - 排序长度
- `descended` - 降序的标志
- `dtype` - `output_data` 和 `input` 的元素的数据类型

注意事项

- `output_data_addr`、`output_idx_addr` 和 `input_addr` 都被 4 整除。
- `dtype` 的有效取值是 `DT_FP32`、`DT_INT32` 和 `DT_UINT32`, `output_idx` 的元素的数据类型是 `INT32`。
- `output_data` 和 `output_idx` 的长度是 `len`, 前 `K` 个数是排序后的结果和对应的索引, `K` 小于等于 `len`。

4.22.3 tpu_hau_sort_specific_index

按升序或降序稳定排序前 `K` 个最小或最大的数, 并输出排序后的索引, 排序前的索引是指定索引。

```
tpu_hau_sort_specific_index(system_addr_t output_data_addr, system_addr_t
                             output_idx_addr, system_addr_t input_data_addr,
                             system_addr_t input_idx_addr, int len, int K, bool
                             descended, data_type_t dtype)
```

$$\text{output_data}(k) = \text{input_data}(i_k) \quad \text{output_idx}(k) = \text{input_idx}(i_k)$$

如果 $\text{input_data}(i_k) = \text{input_data}(i_{k+1})$, 则 $\text{input_idx}(i_k) \leq \text{input_idx}(i_{k+1})$

如果升序, 则

$$\text{input_data}(i_0) \leq \text{input_data}(i_1) \leq \cdots \leq \text{input_data}(i_{K-1}) \leq \cdots \leq \text{input_data}(i_{\text{len}-1})$$

如果降序, 则

$$\text{input_data}(i_0) \geq \text{input_data}(i_1) \geq \cdots \geq \text{input_data}(i_{K-1}) \geq \cdots \geq \text{input_data}(i_{\text{len}-1})$$

其中, $i_0, i_1, \dots, i_{\text{len}-1}$ 互不相同, 是 $0, 1, \dots, \text{len} - 1$ 的重排。

参数

- `output_data_addr` - `output_data` 的地址
- `output_idx_addr` - `output_idx` 的地址
- `input_data_addr` - `input_data` 的地址
- `input_idx_addr` - `input_idx` 的地址
- `len` - `input_data` 和 `input_idx` 的长度
- `K` - 排序长度
- `descended` - 降序的标志
- `dtype` - `output_data` 和 `input` 的元素的 datatype

注意事项

- `output_data_addr`、`output_idx_addr`、`input_data_addr` 和 `input_idx_addr` 都被 4 整除。
- `dtype` 的有效取值是 `DT_FP32`、`DT_INT32` 和 `DT_UINT32`，`output_idx` 和 `input_idx` 的元素的 datatype 是 `INT32`。
- `output_data` 和 `output_idx` 的长度是 `len`，前 `K` 个数是排序后的结果和对应的索引，`K` 小于等于 `len`。

4.22.4 tpu_hau_line_gather

通过 `line` 的索引取值得到输出张量，即 `output = param[index]`。

```
tpu_hau_line_gather(system_addr_t output_addr, system_addr_t param_addr,
                    system_addr_t index_addr, scalar_t C, int line_num, int line_len,
                    int index_len, int start, int end, data_type_t dtype, bool
                    fill_const)
```

$$\text{output}(h, w) = \begin{cases} \text{param}(\text{index}(h) - \text{start}, w) & \text{如果 index}(h) \text{ 是有效索引} \\ C & \text{如果 index}(h) \text{ 是无效索引, fill_const 是 true} \end{cases}$$

参数

- `output_addr` - `output` 的地址
- `param_addr` - `param` 的地址
- `index_addr` - `index` 的地址
- `C` - 常数
- `line_num` - `param` 的 `line` 的数量
- `line_len` - `param` 的 `line` 的长度
- `index_len` - `index` 的长度
- `start` - 有效索引的起始值
- `end` - 有效索引的结束值
- `dtype` - `output` 和 `param` 的元素的 datatype
- `fill_const` - `output` 在无效索引处填 `C` 的标志

注意事项

- `output_addr`、`param_addr` 和 `index_addr` 都被 64 整除。
- `output` 的 shape 是 `[index_len, line_len]`，`param` 的 shape 是 `[line_num, line_len]`，`index` 的 shape 是 `[index_len]`，都是 continuous layout。

- index 的元素的数据类型是 UINT32，有效索引的范围是 [start, end]，start 大于等于 0，start 小于等于 end，end 小于 line_num。
- 如果索引无效，fill_const 是 false，则 output 的对应元素不会被填。