
TPU-KERNEL Technical Reference Mannal

SOPHGO

May 10, 2024

CONTENTS

1	Introduction	2
1.1	Introduction	2
2	TPU Architecture	3
2.1	TPU Architecture	3
2.2	Memory Types	4
3	Programming	5
3.1	TPU Working Mode	5
3.2	TPU Programming	5
3.3	Host Side	7
3.4	Device Side	10
4	TPU API	23
4.1	Basic Definitions	23
4.2	Rounding Mode	26
4.3	Common Functions	28
4.4	Synchronization Functions	31
4.5	Utils Functions	32
4.6	GDMA Functions	35
4.7	Basic BDC Functions	64
4.8	Data Conversion and Rounding Functions	67
4.9	Unary Functions	69
4.10	Binary Functions	80
4.11	Floating Point Binary Functions	104
4.12	Fixed Point Binary Functions	118
4.13	Compare and Select Functions	135
4.14	Floating Point Matrix Functions	139
4.15	Fixed Point Matrix Functions	147
4.16	Floating Point Neural Network Functions	169
4.17	Fixed Point Neural Network Functions	179
4.18	Activation Functions	181
4.19	Scatter and Gather Functions	192
4.20	Special Functions	201
4.21	Quantization Functions	206
4.22	HAU Functions	212



Legal Disclaimer

- Copyright © SOPHGO 2024. All rights reserved.
- No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of SOPHGO .

Notice

- The purchased products, services and features are stipulated by the contract made between SOPHGO and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided “AS IS” without warranties, guarantees or representations of any kind, either express or implied. The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Technical Support

Address Floor 6, Building 1, Yard 9, FengHao East Road, Haidian District, Beijing, 100094,
China

Website <https://www.sophgo.com/>

Email sales@sophgo.com

Phone +86-10-57590723 +86-10-57590724

INTRODUCTION

1.1 Introduction

TPUKernel is an underlying programming interface for SOPHGO deep-learning processor BM1684X. The interface can support the following operations:

1. Call special instructions (such as convolution, pooling, etc.) to accelerate deep learning functions.
 2. Call general instructions (such as matrix multiplication, etc.) to accelerate various algorithms customized by users.
-

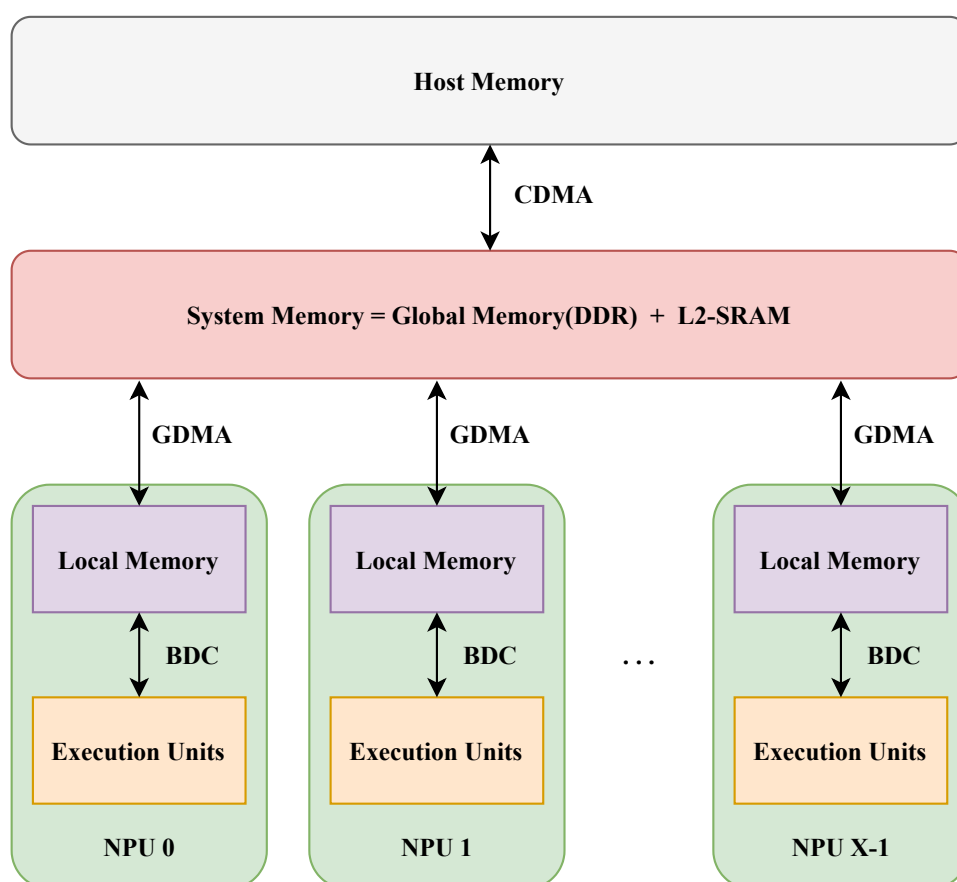
The document contains the following:

1. Introduction to the chip architecture of BM1684X.
 2. The API (Application Programming Interface, Packaged Instruction Set) of BM1684X.
-

TPU ARCHITECTURE

2.1 TPU Architecture

The following is the architecture of SOPHGO deep-learning processor BM1684X.



SOPHGO deep-learning processor has a multi-core architecture. Each core is called a **Neural network Processing Unit (NPU)**. There is an independent memory and many kinds of **Execution Units (EU)** in each NPU.

All NPUs execute instructions in the form of Single Instruction Multiple Data (SIMD). At a certain moment, all NPUs will execute the same instructions, but the data on each NPU is different.

The memory inside each NPU is called local memory, and the EUs can only access local memory. The data needs to be copied from system memory (usually global memory) to NPU local memory by GDMA before it can be accessed by EUs. CDMA, GDMA and BDC can run in parallel.

The calculation acceleration of TPU is usually divided into the following steps:

1. Copy the data from the host-side memory to the TPU' s system memory (global memory).

2. Copy the data from system memory (global memory) to local memory.
 3. Calculate the data from local memory, and return the calculation result to local memory.
 4. Copy the results from local memory back to global memory.
 5. Copy the results from global memory back to the host-side memory.
-

2.2 Memory Types

The TPU contains the following memory types:

- **System Memory**
 - Global Memory: Off-chip memory (DDR).
 - L2-SRAM: On-chip memory, it can be used as intermediate cache.
- **Local Memory** : On-chip memory, it is mainly used to store the data for BDC.

PROGRAMMING

3.1 TPU Working Mode

In SOPHGO deep-learning processor, according to the difference of the main control unit (Host side), the TPU has two different working modes, which are called **PCIe Mode** and **SoC Mode** respectively.

- **PCIe Mode:** The products in PCIe mode are SC series boards. The board is connected to the host server through PCIe interface. The host server act as the main control unit (Host) to control the operation of the board.
 - **SoC Mode:** The products in SoC mode are SE series devices. The inference device contains an 8-core A53 processor as the main control unit (Host) to control the operation of the board.
-

3.2 TPU Programming

TPU is a heterogeneous architecture, which requires the host to send instructions, and the device to receive the instructions and execute them. Therefore, to perform the calculation on TPU, it is necessary to write two parts of the code on host and device respectively:

- **Host:** Host-side code, runs on the host and sends commands to control the TPU.
- **Device:** Device-side code, running on the device, usually perform various instructions of the TPU.

To fit different target devices, the host-side and device-side codes need to be compiled with different compilers.

In **PCIe Mode**,

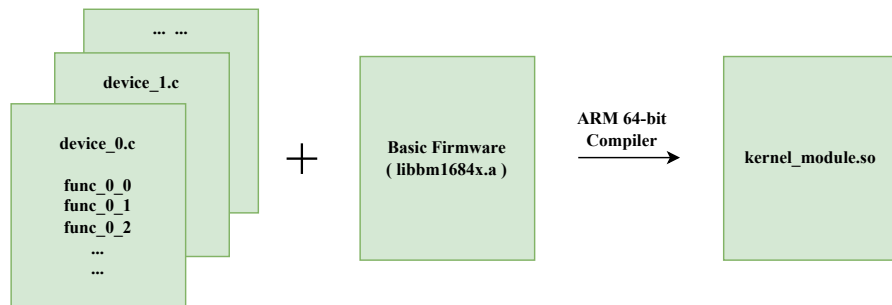
- Host-side code compiler: The host' s C++ compiler used by user.
- Device-side code compiler: Cross compiler runs by on-chip Linux ARM A53 processor (also called none ARM53 processor).

In **SoC Mode**, the code is compiled on x86 platform and runs on ARM A53 platform.

- Host-side code compiler: Cross compiler runs by on-chip Linux ARM A53 processor.
- Device-side code compiler: Cross compiler runs by on-chip Linux ARM A53 processor (also called none ARM53 processor).

The above ARM A53 cross compiler can be downloaded through the script `scripts / prepare_toolchain.sh` in the TPUKernel toolkit.

For device-side code, the compilation process can be seen as the process of updating dynamic link libraries.



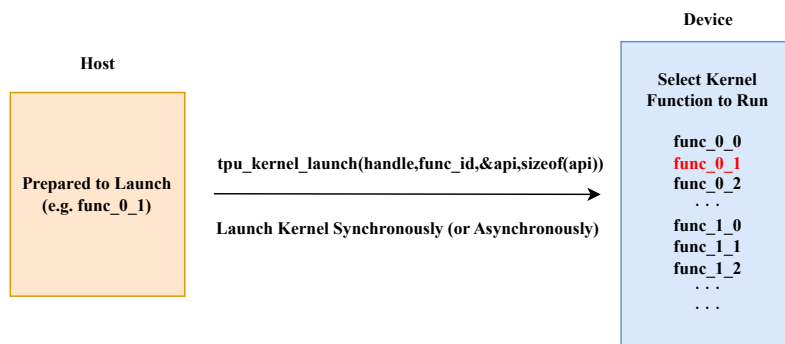
As shown in the above diagram, `:cpp:func:device_*.c` is the device-side code completed by the developer. The device-side code compiler links the device-side code with the original bottom library `libbm1684x.a` to form a complete A53Lite loadable raw dynamic library file.

3.3 Host Side

After completing the update of the above dynamic library, it is only necessary to load the dynamic library, and then send the call instruction from the host to control the TPU to run the specified calculation.

```
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);
```



As shown in the figure above, the device firmware has already registered `func_0_0()` , `func_0_1()` , `func_0_2()` , `func_1_0()` , `func_1_1()` , `func_1_2()` ...and other functions, The host side now sends `func_0_1()`, and the TPU will perform the calculation of `func_0_1()`.

There are two ways for the host to call the TPU for calculation, **asynchronous** and **synchronous**.

3.3.1 Synchronous Mode

Synchronous mode means that after sending the call command, the host will not continue to execute the following host-side code until the calculation on TPU is completed.

If the host calls the following API, it will wait for TPU until all calculation is completed.

```
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.

Parameters

- **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.

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

3.3.2 Asynchronous Mode

Asynchronous mode means that after sending the call command, the host will continue to execute the following host-side code. The TPU and the host run asynchronously.

If the host calls the following API, the TPU will start to calculate, and the host side will continue to execute the following code.

```
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.

Parameters

- **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.

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

The host can achieve synchronization with the TPU through the following API. After calling it, the host will wait for TPU to complete calculation.

```
bm_status_t tpu_kernel_sync(bm_handle_t handle)  
Synchronize the device.
```

Parameters **handle** – Handle of the device.

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

3.4 Device Side

In the previous section, we introduced how to drive the TPU to perform calculation. In this section, we will start to learn how to write device-side code and use TPU to complete the calculations we defined. Before that, we need to understand what basic instructions are defined by TPU. In the **TPU Architecture** section, we have introduced the entire calculation process of the TPU. This process can be divided into three parts:

1. Copy data between host memory and system memory back and forth.
2. Copy data between system memory and local memory back and forth.
3. The TPU perform calculations on the data in local memory.

The first part is related to the host-side. Since the data are saved in host memory, the related API is usually called by host. For the relevant commands on host side, please refer to the previous section.

3.4.1 Command System

· GDMA Command

Operations related to data transfer between system memory and local memory are all done by GDMA commands. GDMA-related commands all start with `tpu_gdma_()`, including data transfer between different local memory and between different system memory. For detailed command descriptions and parameters, please refer to the **TPU API** chapter.

· BDC Command

Operations related to data calculation performed by the TPU can be completed by BDC commands. All BDC commands start with `tpu_bdc_()`.

· HAU Command

Some commands that are not suitable for parallel accelerated computing, including NMS, SORT, etc.

3.4.2 Memory and Data Alignment

· Data Representation

Tensor

In TPU, data is stored as a tensor. **Tensor** is a 4-dimensional array. We use a 4-tuple (N,C,H,W) to describe the shape of a tensor. `Tensor(n, c, h, w)` represents the data element at index of (n, c, h, w).

A tensor in local memory can not be described only by the address, shape and data type, the strides are also necessary. For a 4D tensor with shape (N, C, H, W), there are four relative strides named `N_stride`, `C_stride`, `H_stride` and `W_stride`.

Stride indicates how many elements are separated between elements of the same dimension when tensor is stored in local memory.

For arbitrary n in $[0, N - 1]$, c in $[0, C - 1]$, h in $[0, H - 1]$ and w in $[0, W - 1]$,

- `N_stride`: number of elements from (n, c, h, w) to (n + 1, c, h, w).
- `C_stride`: number of elements from (n, c, h, w) to (n, c + X, h, w), where X is the number of NPUs.

- H_stride: number of elements from (n, c, h, w) to $(n, c, h + 1, w)$.
- W_stride: number of elements from (n, c, h, w) to $(n, c, h, w + 1)$.

Suppose there is a tensor has a shape of $(4, 3, 2, 2)$, each of its elements occupies 1 byte. If its stride in local memory is $(12, 4, 2, 1)$, it will be arranged as shown in the following figure:

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

If the stride is $(24, 4, 2, 1)$, it will be arranged as follows:

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

If the stride is $(24, 8, 4, 2)$, it will be arranged as follows:

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

If the stride is $(24, 8, 4, 1)$, it will be arranged as follows:

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

Data Type

The unit of **Stride** is the number of elements. Different types of data elements have different bytes. The data types in following are supported on BM1684X TPU:

Data Type	Bytes
INT8	1 Bytes
INT16	2 Bytes
INT32	4 Bytes
FP16	2 Bytes
BFP16	2 Bytes
FP32	4 Bytes

Tensor Layout in Global Memory

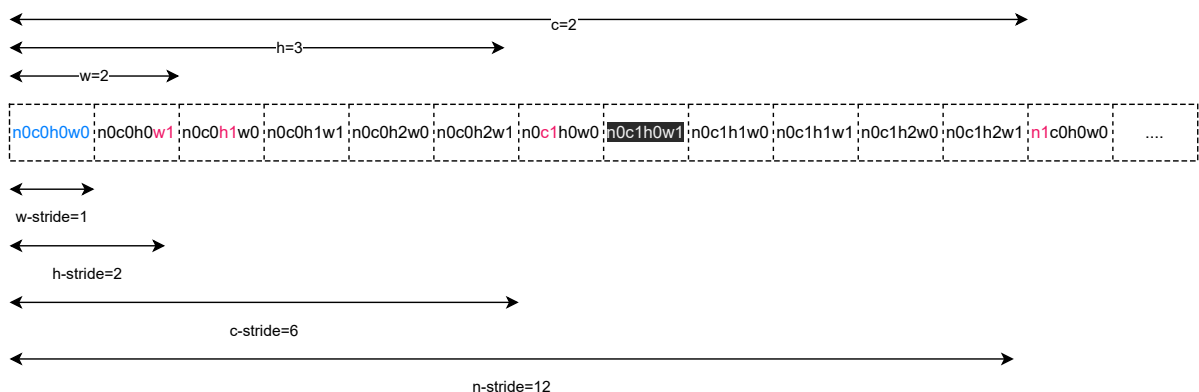
Global memory consists of a piece of DDR memory.

A tensor with shape of (N, C, H, W), arranged in **global memory**. If its stride is:

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

This layout is called **continuous storage**.

Example: A tensor with shape of (N=2, C=2, H=3, W=2) is arranged in global memory as follows:



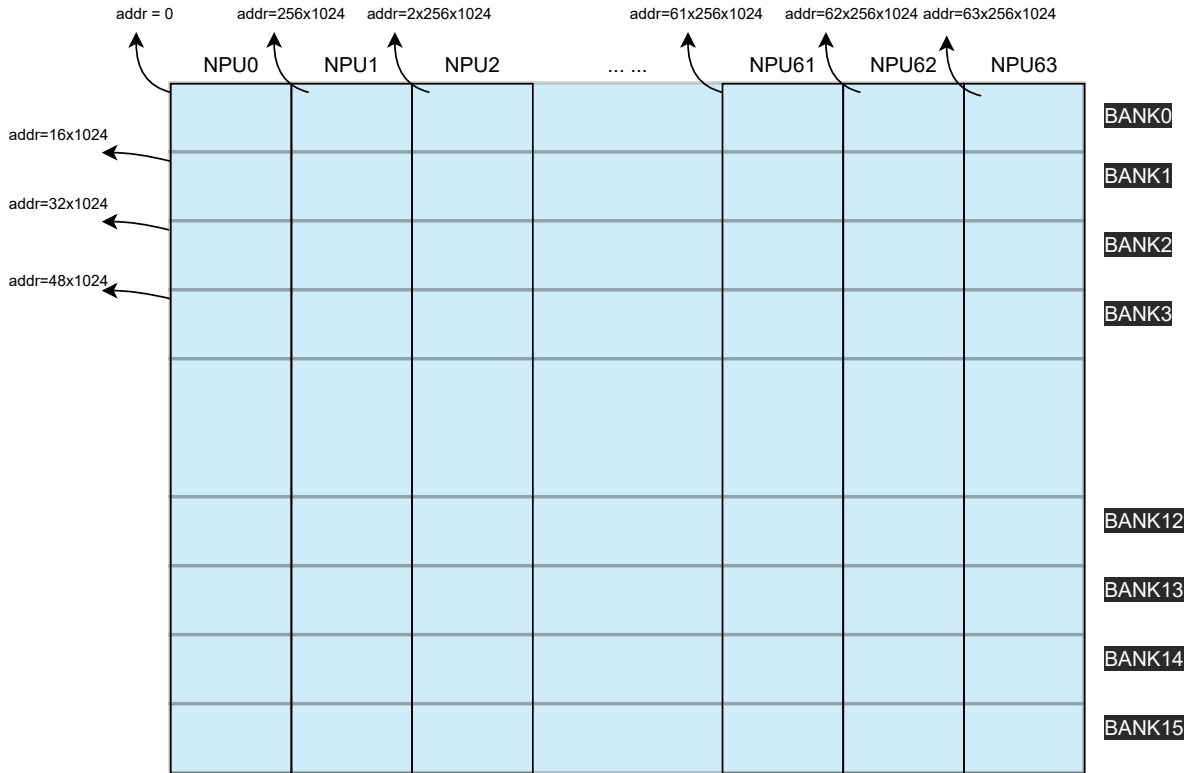
In the figure above, `n0c0h0w0` represents the element at the position of Tensor $(0,0,0,0)$.

Tensor Layout in Local Memory

Composition and Address of Local Memory

The local memory consists of multiple pieces of SRAM (Static Random Access Memory), and each piece of SRAM is called a **bank**. The BM1684X consists of 16 banks in total. 16 banks constitute the whole **local memory**.

The entire local memory is also divided into 64 **lanes** (corresponding to 64 NPUs, and then NPUs are used to refer to lanes). The address assignment is shown in the following figure:



The local memory size of BM1684X is **256KB * 64**, and the address is allocated according to the NPU. Among them, NPU0 has the address of `0~256*1024-1`, NPU1 has the address of `256*1024~2*256*1024-1`, and so on.

Rules for Tensor Layout in Local Memory

The layout of tensors on local memory is different from global memory. The main difference is the layout in the **C dimension**.

For a tensor with shape (N, C, H, W) , it has $N * C$ features, if copied to local memory, the features will be scattered to different NPUs. Denote the feature $(n, c, :, :)$ simply by (n, c) , where n is in $[0, N - 1]$ and c is in $[0, C - 1]$. Let the number of NPUs be X .

Example: A tensor with shape $(N=2, C=3, H=2, W=3)$, stride $(N_stride = 9, C_stride = 9, H_stride = 3, W_stride = 1)$. The layout on local memory is as follows.

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

Another important concept is the number of channels per NPU. **Supposing the tensor starts at NPU Q, the number of channels per NPU is $\text{ceil}((Q + C) / X)$.** The *ceil* function represents rounding up to an integer.

Case 1: Supposing $C = X - 1$ and the tensor starts at NPU 0, the following figure shows how the features are scattered (Note that each block represents a memory area for storing a feature).

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)	

No features are scattered to NPU X - 1, the feature (1, 0) is not in NPU X - 1, instead, it is in NPU 0. It is found that the features (*, c) are in the same NPU for arbitrary fixed c in $[0, C - 1]$. The number of channels per NPU is $\text{ceil}((0 + X - 1) / X) = 1$.

Case 2: Supposing $C = X - 1$ and the tensor starts at NPU 1, the following figure shows how the features are scattered.

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)

No features are scattered to NPU 0, like NPU X - 1 in the previous case. Still, the number of channels per NPU is $\text{ceil}((1 + X - 1) / X) = 1$.

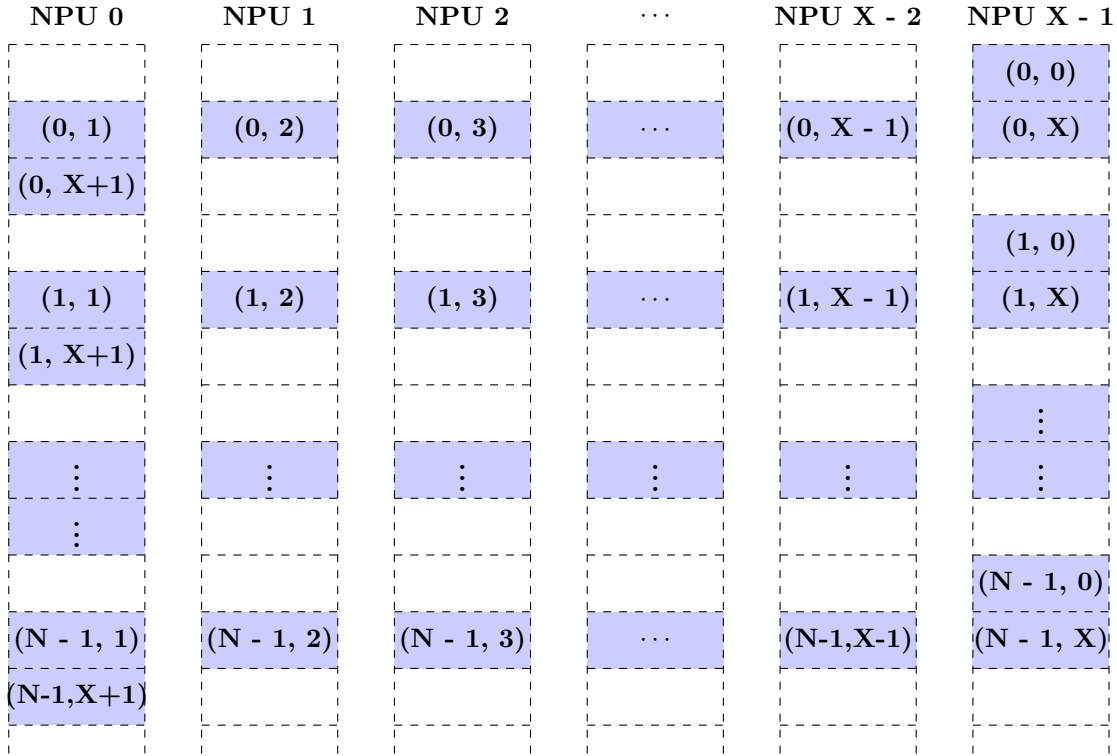
Case 3: Supposing $C = X + 2$ and the tensor starts at NPU 0, the following figure shows how the features are scattered.

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)				

The features $(*, X)$ and $(*, X + 1)$ are scattered to NPU 0 and NPU 1, respectively. In the second block row, there are $X - 2$ empty blocks storing nothing about this tensor. The number of channels per NPU is $\text{ceil}((0 + X + 2) / X) = 2$.

In this case, not only the features $(*, c)$ are in the same NPU, but also $(*, c + X)$ if $c + X$ is in $[0, C - 1]$. The features $(*, c_0)$ and $(*, c_1)$ are in the same NPU if and only if $c_0 \bmod X = c_1 \bmod X$.

Case 4: Supposing $C = X + 2$ and the tensor starts at NPU X - 1, the following figure shows how the features are scattered.



Three block rows are used to store the C features of one batch with X - 1 empty blocks in each the first and third row. The number of channels per NPU is $\text{ceil}((X - 1 + X + 2) / X) = 3$. Such a storage is terrible and causes a lot of waste of local memory.

If reusing the empty blocks to store some other tensors, it should be much careful and know the TPU well, otherwise, the data in these tensors may be destroyed when NPUs work in parallel. Try not to reuse the empty blocks.

Common Layout in Local Memory

The previous chapter describes the basic principles of layout in local memory. Next, we introduce several tensor layouts commonly used in BM1684X.

1. 64-Byte Aligned Layout

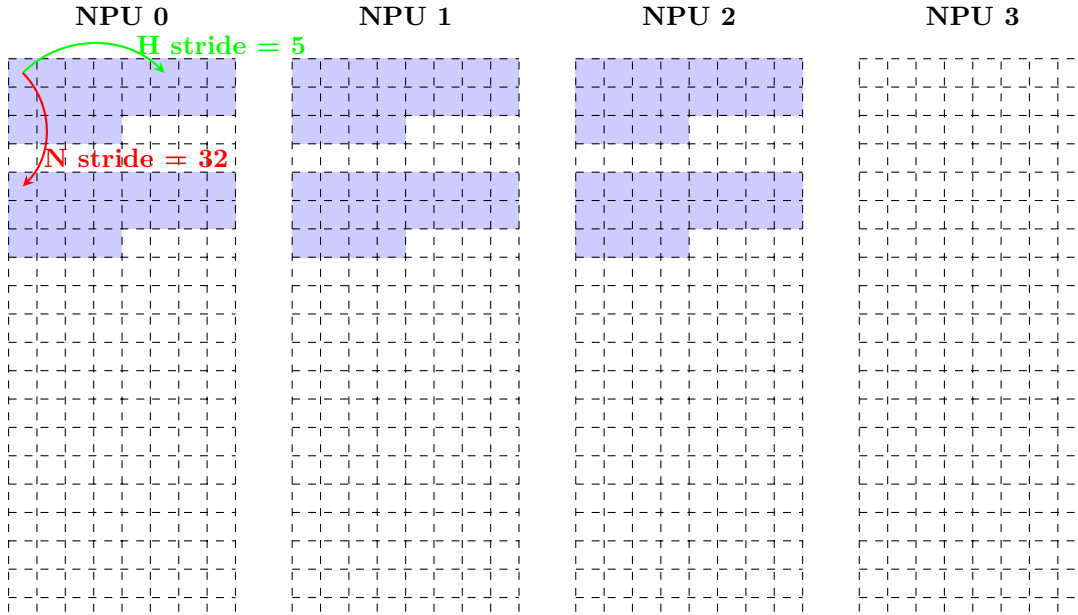
64-byte aligned layout is most commonly used. If a tensor with shape (N, C, H, W) is in the aligned layout, it is required that

- The address of the tensor is divisible by 64.
- The W_stride is 1.
- The H_stride is W.
- The C_stride is $\text{ceil}(H * W / 16) * 16$ if the data type is 32-bit, $\text{ceil}(H * W / 32) * 32$ if the data type is 16-bit, $\text{ceil}(H * W / 64) * 64$ if the data type is 8-bit.
- The N_stride is the C_stride multiply by the number of channels per NPU.

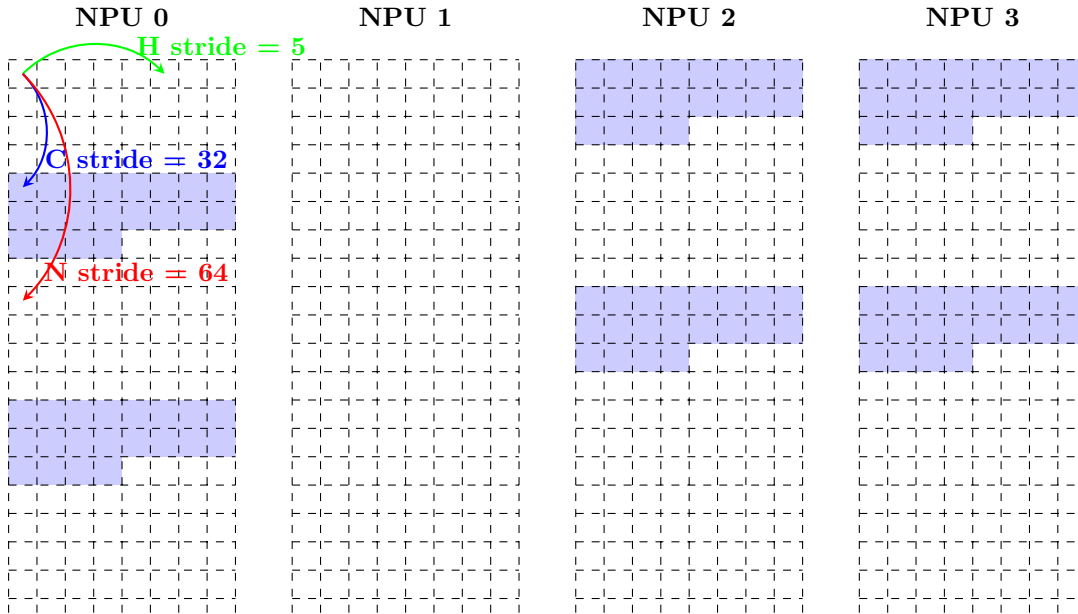
The strides can be obtained by calling `tpu_aligned_stride()`.

To illustrate simply, the number of the NPUs is set to 4.

Given a tensor with shape (N = 2, C = 3, H = 4, W = 5) and data type fp16, the following figure shows how it is in the aligned layout (Note that each block has size of 2 bytes).



Supposing the tensor starts at NPU 2, the following figure shows how the tensor is in the aligned layout.



2. Compact Layout

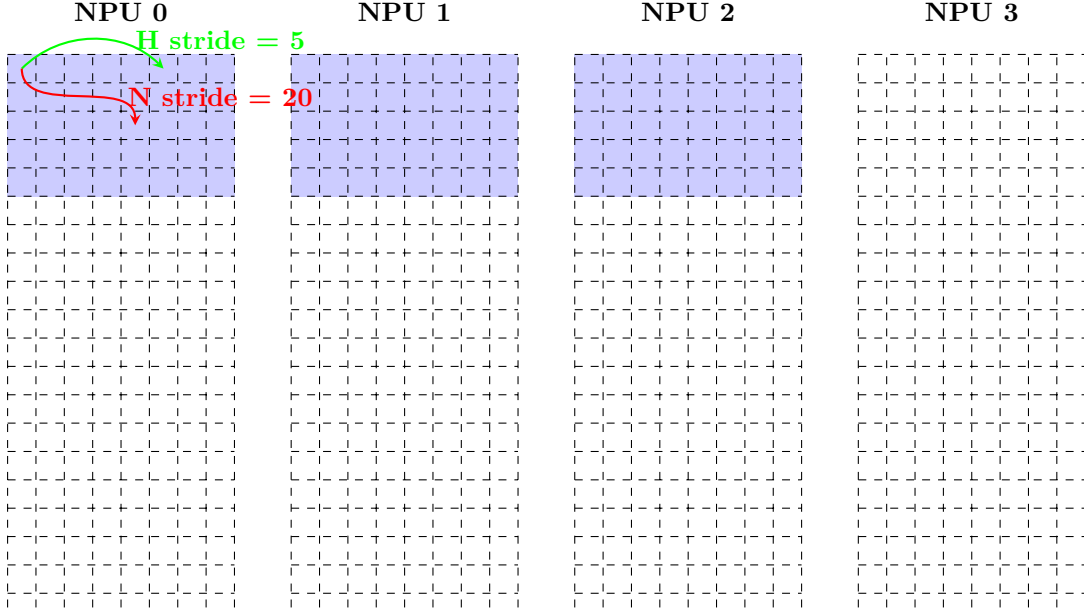
Compact layout is also a commonly used layout. If a tensor with shape (N, C, H, W) is in the compact layout, it is required that

- The address of the tensor is divisible by 4.
- The W_stride is 1.
- The H_stride is W.
- The C_stride is H * W.
- The N_stride is the C_stride multiply by the number of channels per NPU.

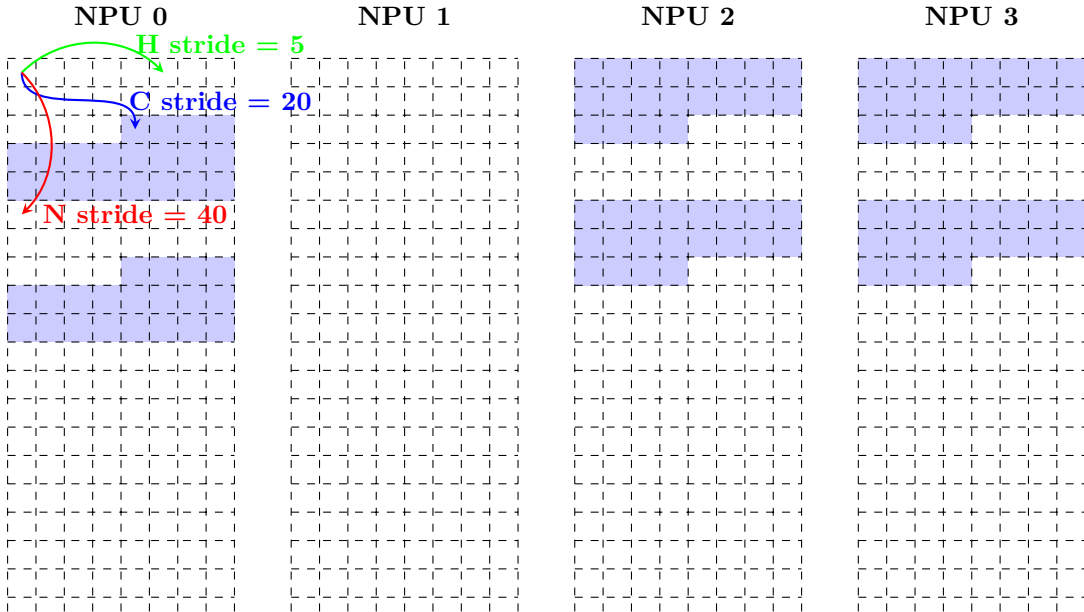
The strides can be obtained by calling `tpu_compact_stride()`.

To illustrate simply, the number of the NPUs is set to 4.

Given a tensor with shape ($N = 2$, $C = 3$, $H = 4$, $W = 5$) and data type fp16, the following figure shows how it is in the compact layout (Note that each block has size of 2 bytes).



Supposing the tensor starts at NPU 2, the following figure shows how the tensor is in the aligned layout.

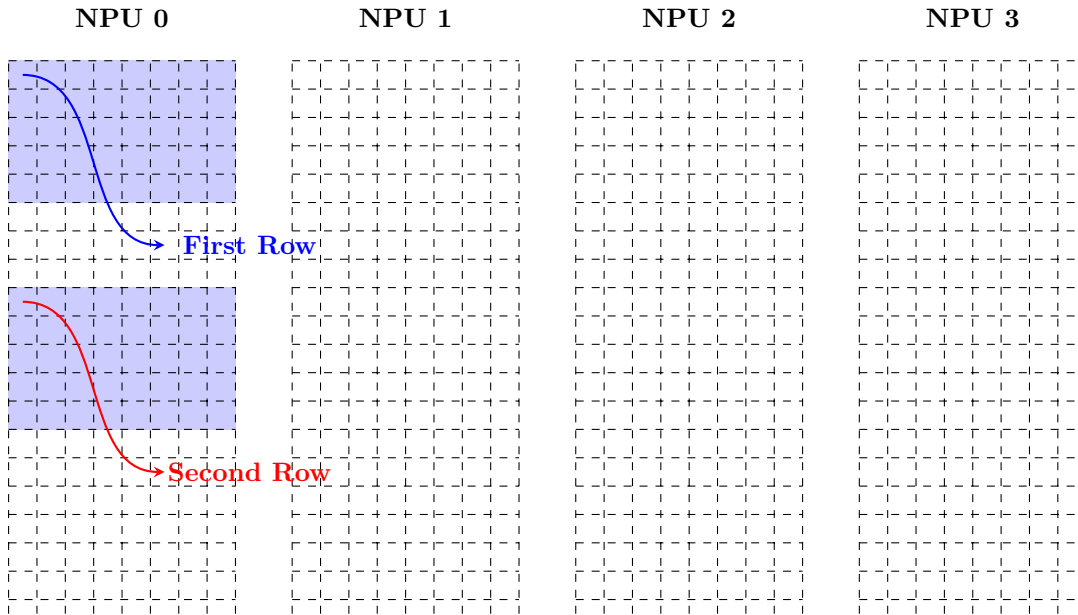


3. Matrix Layout

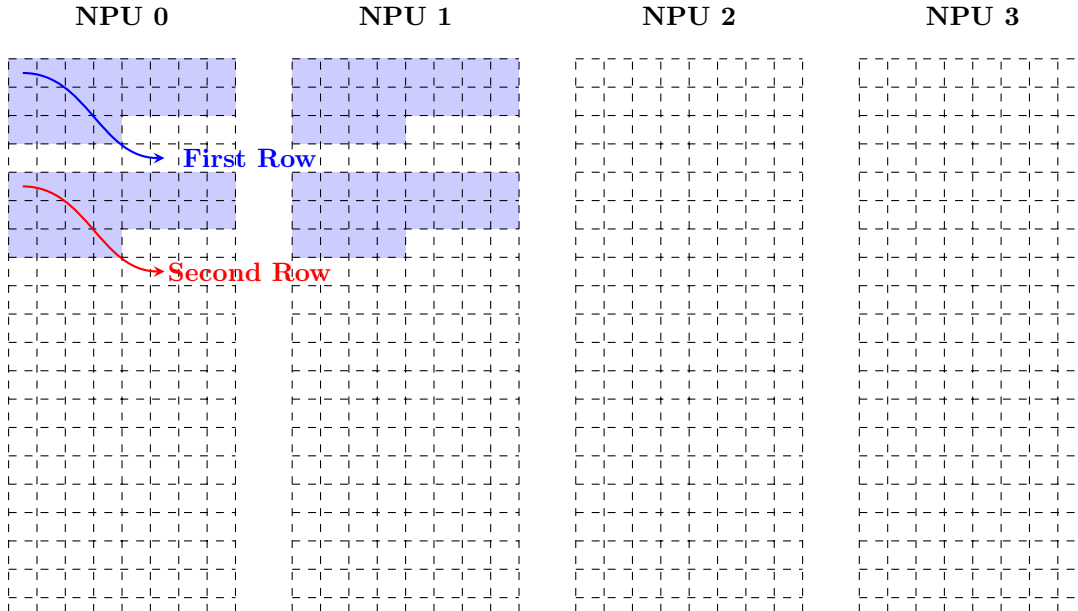
Matrix layout is a layout of matrix in local memory. If a matrix with size n-by-m is in the matrix layout, it could be viewed as a 4D tensor with shape ($N = n$, $C = \text{ceil}(m / w)$, $H = 1$, $W = w$) in the 64-Byte Aligned Layout, where w is in $[1, m]$ and given by user.

To illustrate simply, the number of the NPUs is set to 4.

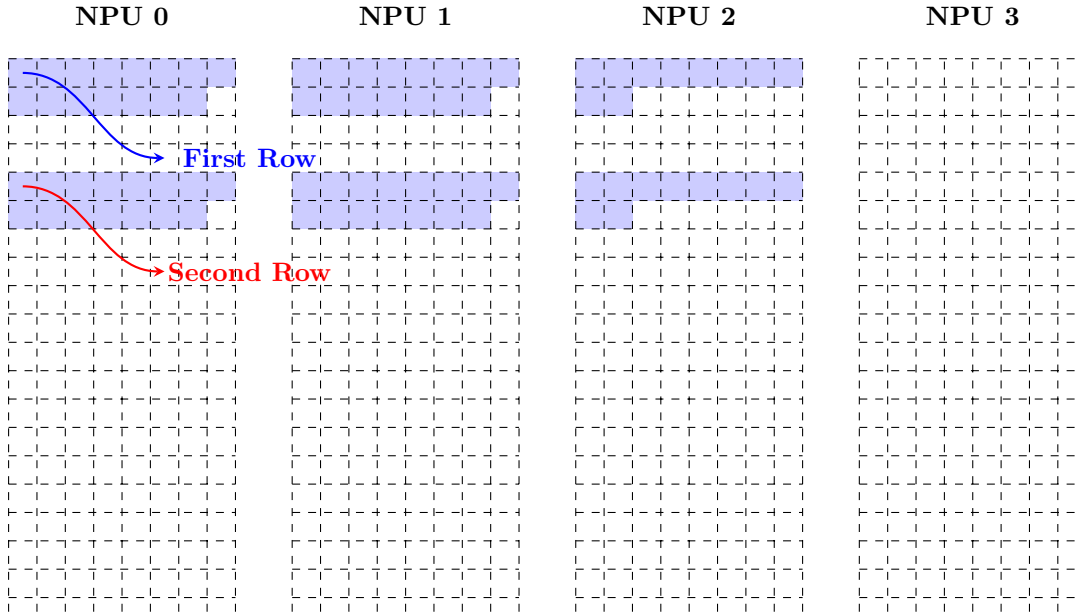
Example 1: Given a matrix with size 2-by-40 and data type fp16, $w = 40$, the following figure shows how it is in the matrix layout (Note that each block has size of 2 bytes).



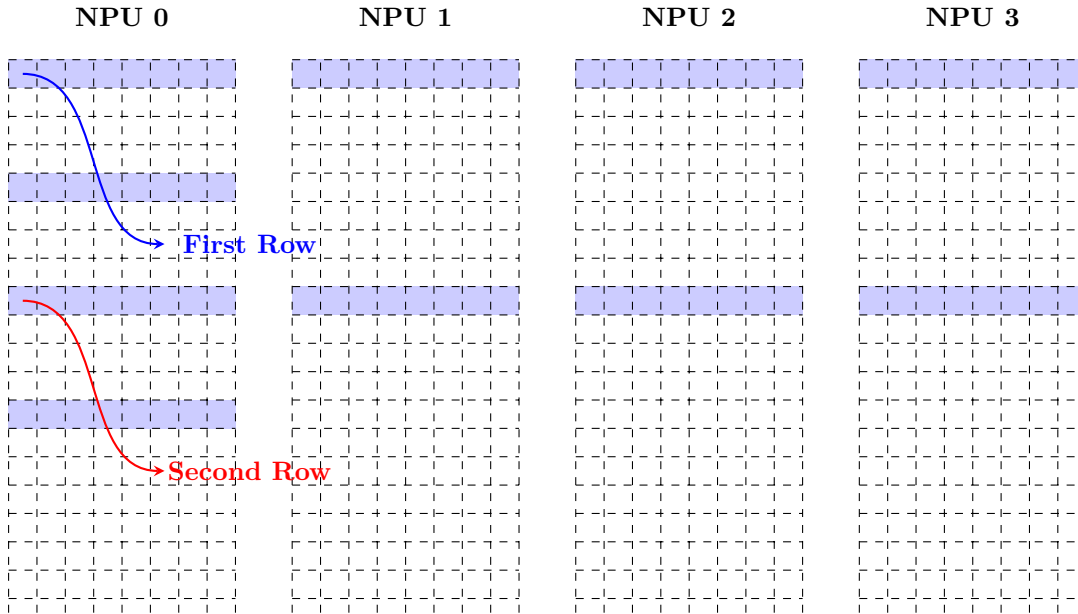
Example 2: Taking $w = 20$, the following figure shows how the matrix is in the matrix layout.



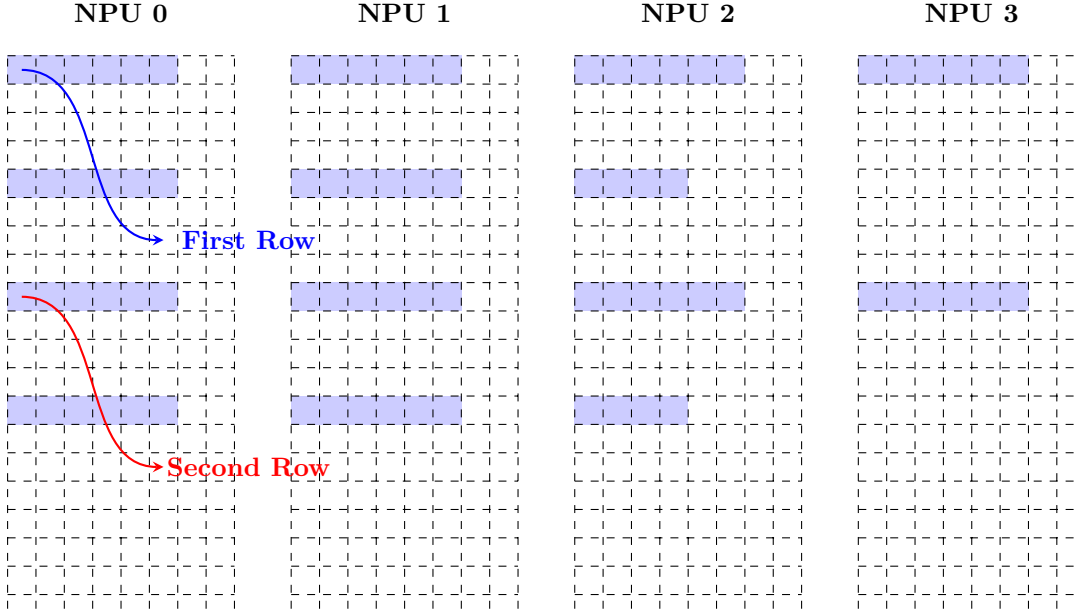
Example 3: Taking $w = 15$, the following figure shows how the matrix is in the matrix layout.



Example 4: Keeping decreasing w and taking $w = 8$, the following figure shows how the matrix is in the matrix layout.



Example 5: Taking $w = 6$, the following figure shows how the matrix is in the matrix layout.



4. Vector Layout

Vector layout is a layout of vector in local memory. If a vector with size 1-by-m is in the vector layout, it could be viewed as a 4D tensor with shape ($N = 1$, $C = \text{ceil}(m / w)$, $H = 1$, $W = w$) in the 64-Byte Aligned Layout, where w is in $[1, m]$ and given by user.

5. Line 64-Byte Aligned Layout

Line 64-byte aligned layout is a layout of tensor in local memory. If a tensor with shape (N , C , H , W) is in this layout, it is required that

- The address of the tensor is divisable by 64.
- The W_stride is 1.
- The H_stride is $\text{ceil}(W / 16) * 16$ if the data type is 32-bit, $\text{ceil}(W / 32) * 32$ if the data type is 16-bit, $\text{ceil}(W / 64) * 64$ if the data type is 8-bit.
- The C_stride is $H * H_stride$.
- The N_stride is the C_stride multiply by the number of channels per NPU.

The strides can be obtained by calling `tpu_line_aligned_stride()`.

6. 64IC/32IC Layout

64IC/32IC layout is a special layout of the convolution kernel in local memory. It is only used for the convolution. The kernel is stored in 64IC layout with the data type of INT8, and stored in 32IC layout with the data type of FP16 or BFP16.

Suppose the shape of the convolution kernel is (ic , oc , kh , kw), which represent the input channel, the output channel, the height of the kernel, and the width of the kernel, respectively.

If the kernel is in 64IC layout, it is required that

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

Among them, each 64 channels are stored as a group, the stride of each group is $64 * kw * kh$.

If the kernel is in 32IC layout, it is required that

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

Among them, each 32 channels are stored as a group, the stride of each group is $32 * kw * kh$.

4.1 Basic Definitions

- Type of address.

```
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
```

- Class of dimension.

```
class dim4
```

```
    int n
```

```
    int c
```

```
    int h
```

```
    int w
```

```
class dim2
```

```
    int h
```

```
    int w
```

- Type of data.
-

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

-
- Enumeration of rounding modes.

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

-
- Half-precision floating-point format.

union **float16**

unsigned short **bits**

- Brain floating-point format.

```
union bfloat16

unsigned short bits
```

- Union of scalar.

```
union scalar_t

char s8

unsigned char u8

short s16

unsigned short u16

float16 f16

bfloat16 bf16

int s32

unsigned int u32

float f32
```

- Enumeration of variable type.

```
enum var_type_t

enumerator TENSOR

enumerator SCALAR

enumerator VECTOR
```

- Union of variable content.

```
union var_context_t

scalar_t scalar_t

local_addr_t addr
```

- Class of variable. If *type* is *TENSOR* or *VECTOR*, then *context.addr* is valid, otherwise *context.scalar* is valid.

```
class variable_t
```

```
var_type_t type
```

```
var_context_t context
```

- Number of NPU.

```
NPU_NUM
```

- Size of local memory on each NPU.

```
LOCAL_MEM_SIZE
```

- Basic mathematical operations.

```
DIV_UP
```

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

```
ALIGN
```

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

4.2 Rounding Mode

Rounding means replacing a number with an approximate value that has a shorter, simpler, or more explicit representation. For given x , if the rounded result is y , there are the following rounding modes for selection.

4.2.1 Half to Even

Rounding: when the decimal value is 0.5, round to the nearest even number. The enumeration value is *RM_HALF_TO_EVEN*.

4.2.2 Half Away From Zero

Rounding: positive numbers are close to positive infinity, negative numbers are close to negative infinity. The enumeration value is *RM_HALF_AWAY_FROM_ZERO*. The formula is

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

4.2.3 Towards Zero

Unconditionally round off decimals, close to zero point. The enumeration value is `RM_TOWARDS_ZERO`. The formula is

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

4.2.4 Down

Rounds toward negative infinity. The enumeration value is `RM_DOWN`. The formula is

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

4.2.5 Up

Rounds toward positive infinity. The enumeration value is `RM_UP`. The formula is

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

4.2.6 Half Up

Rounded to the nearest positive infinity. The enumeration value is `RM_HALF_UP`. The formula is

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

4.2.7 Half Down

Rounds to the nearest negative infinity. The enumeration value is `RM_HALF_DOWN`. The formula is

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

4.2.8 Example

The following table lists the correspondence between x and y under different rounding modes.

	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 Common Functions

4.3.1 tpu_initialize

Initialize device before calling GDMA and BDC functions.

```
void tpu_initialize()
```

4.3.2 tpu_poll

Synchronize device to make all the previous GDMA and BDC functions done.

```
void tpu_poll()
```

Remarks

- Before calling this function, the parallel mode is required to be inactive.

4.3.3 tpu_parallel_start

Start the parallel mode.

```
void tpu_parallel_start()
```

Remarks

- Before calling this function, the parallel mode is required to be inactive.

4.3.4 tpu_parallel_end

End the parallel mode.

```
void tpu_parallel_end()
```

Remarks

- Before calling this function, the parallel mode is required to be active.

4.3.5 tpu_is_parallel_state

Get the flag of the current parallel mode.

```
bool tpu_is_parallel_state()
```

Returns Flag of the current parallel mode, true means active, otherwise, inactive.

4.3.6 tpu_npu_num

Get the number of NPUs in each TPU.

```
int tpu_npu_num()
```

Returns Number of NPUs.

4.3.7 tpu_bank_num

Get the number of banks in local memory.

```
int tpu_bank_num()
```

Returns Number of banks.

4.3.8 tpu_eu_num

Get the number of EUs of different data types.

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

Parameters *dtype* – Data type.

Returns Number of EUs.

4.3.9 tpu_local_mem_size_per_npu

Get the size in bytes of local memory in each NPU.

```
int tpu_local_mem_size_per_npu()
```

Returns Size of local memory per NPU.

4.3.10 tpu_l2_sram_size

Get the size in bytes of L2-SRAM.

```
int tpu_l2_sram_size()
```

Returns Size of L2-SRAM.

4.3.11 tpu_l2_sram_get_start_addr

the start address of L2-SRAM.

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

Returns Start address of L2-SRAM.

4.3.12 tpu_local_mem_get_start_addr

the start address of first local memory.

```
unsigned int tpu_local_mem_get_start_addr()
```

Returns Start address of first local memory.

4.3.13 tpu_global_mem_addr

Get the pointer through the address of global memory.

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

Parameters `addr` – Address of global memory.

Returns Pointer to global memory.

4.3.14 tpu_local_mem_addr

Get the pointer through the index of NPU and address of local memory.

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

Parameters

- `start_idx` – Index of the NPU.
- `addr` – Address of local memory.

Returns Pointer to local memory.

Remarks

- Value range of `start_idx` is $[0, \text{NPU_NUM} - 1]$.
- Value range of `addr` is $[0, \text{LOCAL_MEM_SIZE} - 1]$.

4.3.15 tpu_local_mem_addr_unified

Get the pointer through the address of the unified local memory.

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

Parameters `addr` – Address of local memory.

Returns Pointer to local memory.

Remarks

- Value range of `addr` is $[0, \text{NPU_NUM} * \text{LOCAL_MEM_SIZE} - 1]$.
- Equivalent to `tpu_local_mem_addr (tpu_npu_index(addr) , addr % LOCAL_MEM_SIZE)`.

4.3.16 tpu_l2_sram_addr

Get the pointer through the address of L2-SRAM.

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

Parameters *addr* – Address of L2-SRAM.

Returns Pointer to L2-SRAM.

4.3.17 tpu_flush_cache

Write back the data from cache to DDR.

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

Parameters

- **address** – DDR' s address, 64-bytes aligned.
- **size** – Data size, it should be any multiple of 64 bytes.

4.3.18 tpu_invalidate_cache

Invalidate cache, read data from DDR directly.

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

Parameters

- **address** – DDR' s address, 64-bytes aligned.
- **size** – Data size, it should be any multiple of 64 bytes.

4.4 Synchronization Functions

4.4.1 tpu_bdc_send_msg

BDC engine sends message to other engines at the given message index, used for data synchronization.

Remarks

- BMB1684X does not support this function.
- Maximum value for *msg_id* is 127.
- Maximum value for *wait_cnt* is 7.

4.4.2 tpu_bdc_wait_msg

BDC engine waits message from other engines at given message index, it can execute command only if received message

Remarks

- BM1684X does not support this function.
- Maximum value for *msg_id* is 127.
- Maximum value for *send_cnt* is 7.

4.4.3 tpu_gdma_send_msg

GDMA engine sends message to other engines at the given message index, used for data synchronization.

Remarks

- BMB1684X does not support this function.
- Maximum value for *msg_id* is 127.
- Maximum value for *wait_cnt* is 7.

4.4.4 tpu_gdma_wait_msg

GDMA engine waits message from other engines at given message index, it can execute command only if received message.

Remarks

- BM1684X does not support this function.
- Maximum value for *msg_id* is 127.
- Maximum value for *send_cnt* is 7.

4.4.5 tpu_hau_send_msg

HAU engine sends message to other engines at the given message index, used for data synchronization.

Remarks

- BMB1684X does not support this function.
- Maximum value for *msg_id* is 127.
- Maximum value for *wait_cnt* is 7.

4.4.6 tpu_hau_wait_msg

HAU engine waits message from other engines at given message index, it can execute command only if received message.

Remarks

- BM1684X does not support this function.
- Maximum value for *msg_id* is 127.
- Maximum value for *send_cnt* is 7.

4.5 Utils Functions

4.5.1 tpu_data_type_size

Get the size of a data type.

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

**** *DT_INT4/DT_UINT4* is not supported, which will cause an assert**

4.5.2 tpu_data_type_bits

Get the bit width of a data type.

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

4.5.3 tpu_npu_index

Get the index of NPU by local address.

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

4.5.4 tpu_bank_index

Get the index of bank by local address.

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

4.5.5 tpu_channle_num_per_npu

Calculate the number of channels in each NPU.

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

Parameters

- **start_idx** – Index of the NPU where the tensor starts.
- **num_channels** – Number of channels of the tensor.

Returns Number of channels per NPU.

4.5.6 tpu_aligned_feature_size

Calculate the memory size of a matrix in the 64-byte aligned layout.

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

Parameters

- **h** – Height h of a 2D vector.
- **w** – Width w of a 2D vector.
- **dtype** – Type of data.

4.5.7 tpu_aligned_stride

Calculate strides of the tensor in the 64-byte aligned layout.

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

Parameters

- **stride** – Pointer to the stride of the tensor.
- **start_idx** – Index of the NPU where the tensor starts.
- **shape** – Pointer to the shape of the tensor.
- **dtype** – Type of data.

4.5.8 tpu_compact_stride

Calculate strides of the tensor in the compact layout.

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

Parameters

- **stride** – Pointer to the stride of the tensor.
- **start_idx** – Index of the NPU where the tensor starts.
- **shape** – Pointer to the shape of the tensor.

4.5.9 tpu_line_aligned_stride

Calculate strides of the tensor in the Line 64-byte aligned layout.

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

Parameters

- **stride** – Pointer to the stride of the tensor.
- **start_idx** – Index of the NPU where the tensor starts.
- **shape** – Pointer to the shape of the tensor.
- **dtype** – Type of data.

4.5.10 tpu_continuous_stride

Calculate strides of the tensor in the continuous layout.

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

Parameters

- **stride** – Pointer to the stride of the tensor.
- **shape** – Pointer to the shape of the tensor.
- **dtype** – Type of data.

4.6 GDMA Functions

4.6.1 tpu_gdma_general_cpy_S2L

Copy tensor from system memory to 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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor in local memory.
- **src_addr** – Address of the source tensor in system memory.
- **dst_shape** – Pointer to the shape of the destination tensor.
- **src_shape** – Pointer to the shape of the source tensor.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dtype** – Data type of source and destination tensor.

注意事项

- Number of elements of **dst_shape** equals to that of **src_shape**.
- If **dst_stride** is NULL, the destination tensor is in 64-byte aligned layout, otherwise in free layout.
- If **src_stride** is NULL, the source tensor is in continuous layout, otherwise in free layout.
- **dst_stride->w** and **src_stride->w** less than or equal to $128 / \text{tpu_data_type_size}(\text{dtype})$.

4.6.2 tpu_gdma_general_cpy_L2S

Copy tensor from local memory to 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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor in system memory.
- **src_addr** – Address of the source tensor in local memory.
- **dst_shape** – Pointer to the shape of the destination tensor.
- **src_shape** – Pointer to the shape of the source tensor.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.

- **dtype** – Data type of source and destination tensor.

注意事项

- Number of elements of **dst_shape** equals to that of **src_shape**.
- If **dst_stride** is NULL, the destination tensor is in continuous layout, otherwise in free layout.
- If **src_stride** is NULL, the source tensor is in 64-byte aligned layout, otherwise in free layout.
- **dst_stride->w** and **src_stride->w** less than or equal to $128 / \text{tpu_data_type_size}(\text{dtype})$.

4.6.3 tpu_gdma_cpy_S2L

Copy tensor from system memory to 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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor in local memory.
- **src_addr** – Address of the source tensor in system memory.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dtype** – Data type of source and destination tensor.

Remarks

- If **dst_stride** is NULL, the destination tensor is in 64-byte aligned layout, otherwise in free layout.
- If **src_stride** is NULL, the source tensor is in continuous layout, otherwise in free layout.
- **dst_stride->w** and **src_stride->w** less than or equal to $128 / \text{tpu_data_type_size}(\text{dtype})$.

4.6.4 tpu_gdma_cpy_L2S

Copy tensor from local memory to 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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor in system memory.

- **src_addr** – Address of the source tensor in local memory.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dtype** – Data type of source and destination tensor.

Remarks

- If **dst_stride** is NULL, the destination tensor is in continuous layout, otherwise in free layout.
- If **src_stride** is NULL, the source tensor is in 64-byte aligned layout, otherwise in free layout.
- **dst_stride->w** and **src_stride->w** less than or equal to $128 / \text{tpu_data_type_size}(\text{dtype})$.

4.6.5 tpu_gdma_cpy_L2L

Copy tensor from local memory to 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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor in local memory.
- **src_addr** – Address of the source tensor in local memory.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dtype** – Data type of source and destination tensor.

Remarks

- If **dst_stride** is NULL, the destination tensor is in 64-byte aligned layout, otherwise in free layout.
- If **src_stride** is NULL, the source tensor is in 64-byte aligned layout, otherwise in free layout.
- **dst_stride->w** and **src_stride->w** less than or equal to $128 / \text{tpu_data_type_size}(\text{dtype})$.

4.6.6 tpu_gdma_cpy_S2S

Copy tensor from system memory to 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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor in system memory.
- **src_addr** – Address of the source tensor in system memory.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dtype** – Data type of source and destination tensor.

Remarks

- If **dst_stride** is NULL, the destination tensor is in continuous layout, otherwise in free layout.
- If **src_stride** is NULL, the source tensor is in continuous layout, otherwise in free layout.
- **dst_stride->w** and **src_stride->w** less than or equal to `128 / tpu_data_type_size(dtype)`.

4.6.7 tpu_gdma_cpy_nc_trans_S2L

Copy tensor from system memory to local memory, with N and C dimensions transposed.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor in local memory.
- **src_addr** – Address of the source tensor in system memory.
- **dst_shape** – Pointer to the shape of the destination tensor.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dtype** – Data type of source and destination tensor.

Remarks

- The shape of the source tensor is [**dst_shape->c**, **dst_shape->n**, **dst_shape->h**, **dst_shape->w**].
- If **dst_stride** is NULL, the destination tensor is in 64-byte aligned layout, otherwise in free layout.

- If `src_stride` is NULL, the source tensor is in continuous layout, otherwise in free layout.
- `dst_stride->w` and `src_stride->w` less than or equal to $128 / \text{tpu_data_type_size}(\text{dtype})$.

4.6.8 tpu_gdma_cpy_nc_trans_L2S

Copy tensor from local memory to system memory, with N and C dimensions transposed.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor in system memory.
- **src_addr** – Address of the source tensor in local memory.
- **dst_shape** – Pointer to the shape of the destination tensor.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dtype** – Data type of source and destination tensor.

Remarks

- The shape of the source tensor is [`dst_shape->c`, `dst_shape->n`, `dst_shape->h`, `dst_shape->w`].
- If `dst_stride` is NULL, the destination tensor is in continuous layout, otherwise in free layout.
- If `src_stride` is NULL, the source tensor is in 64-byte aligned layout, otherwise in free layout.
- `dst_stride->w` and `src_stride->w` less than or equal to $128 / \text{tpu_data_type_size}(\text{dtype})$.

4.6.9 tpu_gdma_cpy_nc_trans_L2L

Copy tensor from local memory to local memory, with N and C dimensions transposed.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor in local memory.
- **src_addr** – Address of the source tensor in local memory.
- **dst_shape** – Pointer to the shape of the destination tensor.
- **dst_stride** – Pointer to the stride of the destination tensor.

- **src_stride** – Pointer to the stride of the source tensor.
- **dtype** – Data type of source and destination tensor.

Remarks

- The shape of the source tensor is [dst_shape->c, dst_shape->n, dst_shape->h, dst_shape->w].
- If dst_stride is NULL, the destination tensor is in 64-byte aligned layout, otherwise in free layout.
- If src_stride is NULL, the source tensor is in 64-byte aligned layout, otherwise in free layout.
- dst_stride->w and src_stride->w less than or equal to 128 / *tpu_data_type_size(dtype)*.

4.6.10 tpu_gdma_cpy_nc_trans_S2S

Copy tensor from system memory to system memory, with N and C dimensions transposed.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor in system memory.
- **src_addr** – Address of the source tensor in system memory.
- **dst_shape** – Pointer to the shape of the destination tensor.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dtype** – Data type of source and destination tensor.

Remarks

- The shape of the source tensor is [dst_shape->c, dst_shape->n, dst_shape->h, dst_shape->w].
- If dst_stride is NULL, the destination tensor is in continuous layout, otherwise in free layout.
- If src_stride is NULL, the source tensor is in continuous layout, otherwise in free layout.
- dst_stride->w and src_stride->w less than or equal to 128 / *tpu_data_type_size(dtype)*.

4.6.11 tpu_gdma_cpy_cw_trans_S2L

Copy tensor from system memory to local memory, with C and W dimensions transposed.

```
void tpu_gdma_cpy_cw_trans_S2L(local_addr_t dst_addr, system_addr_t src_addr,
                               const dim4_t *dst_shape, const dim4_t *dst_stride, const
                               dim4_t *src_stride, data_type_t dtype)
```

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

Parameters

- **dst_addr** – Address of the destination tensor in local memory.
- **src_addr** – Address of the source tensor in system memory.
- **dst_shape** – Pointer to the shape of the destination tensor.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dtype** – Data type of source and destination tensor.

Remarks

- The shape of the source tensor is [dst_shape->n, dst_shape->w, dst_shape->h, dst_shape->c].
- If dst_stride is NULL, the destination tensor is in 64-byte aligned layout, otherwise in free layout.
- If src_stride is NULL, the source tensor is in continuous layout, otherwise in free layout.
- dst_stride->w and src_stride->w are only allowed to be 1.

4.6.12 tpu_gdma_cpy_cw_trans_L2S

Copy tensor from local memory to system memory, with C and W dimensions transposed.

```
void tpu_gdma_cpy_cw_trans_L2S(system_addr_t dst_addr, local_addr_t src_addr,
                               const dim4_t *dst_shape, const dim4_t *dst_stride, const
                               dim4_t *src_stride, data_type_t dtype)
```

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

Parameters

- **dst_addr** – Address of the destination tensor in system memory.
- **src_addr** – Address of the source tensor in local memory.
- **dst_shape** – Pointer to the shape of the destination tensor.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dtype** – Data type of source and destination tensor.

Remarks

- The shape of the source tensor is [dst_shape->n, dst_shape->w, dst_shape->h, dst_shape->c].

- If `dst_stride` is NULL, the destination tensor is in continuous layout, otherwise in free layout.
- If `src_stride` is NULL, the source tensor is in 64-byte aligned layout, otherwise in free layout.
- `dst_stride->w` and `src_stride->w` are only allowed to be 1.

4.6.13 tpu_gdma_cpy_cw_trans_L2L

Copy tensor from local memory to local memory, with C and W dimensions transposed.

```
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)
```

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

Parameters

- `dst_addr` – Address of the destination tensor in local memory.
- `src_addr` – Address of the source tensor in local memory.
- `dst_shape` – Pointer to the shape of the destination tensor.
- `dst_stride` – Pointer to the stride of the destination tensor.
- `src_stride` – Pointer to the stride of the source tensor.
- `dtype` – Data type of source and destination tensor.

Remarks

- The shape of the source tensor is [`dst_shape->n`, `dst_shape->w`, `dst_shape->h`, `dst_shape->c`].
- If `dst_stride` is NULL, the destination tensor is in 64-byte aligned layout, otherwise in free layout.
- If `src_stride` is NULL, the source tensor is in 64-byte aligned layout, otherwise in free layout.
- `dst_stride->w` and `src_stride->w` are only allowed to be 1.

4.6.14 tpu_gdma_cpy_cw_trans_S2S

Copy tensor from system memory to system memory, with C and W dimensions transposed.

```
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)
```

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

Parameters

- `dst_addr` – Address of the destination tensor in system memory.
- `src_addr` – Address of the source tensor in system memory.
- `dst_shape` – Pointer to the shape of the destination tensor.
- `dst_stride` – Pointer to the stride of the destination tensor.

- **src_stride** – Pointer to the stride of the source tensor.
- **dtype** – Data type of source and destination tensor.

Remarks

- The shape of the source tensor is [dst_shape->n, dst_shape->w, dst_shape->h, dst_shape->c].
- If dst_stride is NULL, the destination tensor is in continuous layout, otherwise in free layout.
- If src_stride is NULL, the source tensor is in continuous layout, otherwise in free layout.
- dst_stride->w and src_stride->w are only allowed to be 1.

4.6.15 tpu_gdma_mask_select_L2S

Filter the tensors in local memory by mask and copy them to global memory.

```
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)
```

Parameters

- **dst_addr** – Address of the destination tensor in global memory.
- **src_addr** – Address of the source tensor in local memory.
- **mask_addr** – Address of the masks in global memory or local memory.
- **mask_in_lmem** – Flag of mask in local memory.
- **shape** – Pointer to the shape of source tensor or mask.
- **data_dtype** – Data type of source and destination tensor.
- **mask_dtype** – Data type of the mask.

Remarks

- filter_num after mask_select can be obtained by tpu_gdma_get_filter_num(). tpu_gdma_get_filter_num() has no input parameters and its return data type is *DT_UINT32*.

4.6.16 tpu_gdma_mask_select_S2S

Filter the tensors in global memory by mask and copy them to global memory.

```
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)
```

Parameters

- **dst_addr** – Address of the destination tensor in global memory.
- **src_addr** – Address of the source tensor in global memory.
- **mask_addr** – Address of the masks in global memory or local memory.
- **mask_in_lmem** – Flag of mask in local memory

- **shape** – Pointer to the shape of source tensor or mask.
- **data_dtype** – Data type of source and destination tensor.
- **mask_dtype** – Data type of the mask.

Remarks

- filter_num after mask_select can be obtained by `tpu_gdma_get_filter_num()`. `tpu_gdma_get_filter_num()` has no input parameters and its return data type is `DT_UINT32`.

4.6.17 tpu_gdma_nonzero_L2S

Output the indices of the non-zero elements of the source tensor in local memory to global memory.

```
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)
```

Parameters

- **dst_addr** – Address of the destination tensor in global memory.
- **src_addr** – Address of the source tensor in local memory.
- **shape** – Pointer to the shape of the source tensor.
- **data_type** – Data type of source tensor.
- **base_idx** – The starting index of destination tensor.

Remarks

- The destination tensor is in 64-byte aligned layout. The source tensor is in compact layout.
- The number of indices can be obtained by `tpu_gdma_get_filter_num()`. `tpu_gdma_get_filter_num()` has no input parameters and its return data type is `DT_UINT32`.

4.6.18 tpu_gdma_nonzero_S2S

Output the indices of the non-zero elements of the source tensor in global memory to global memory.

```
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)
```

Parameters

- **dst_addr** – Address of the destination tensor in global memory.
- **src_addr** – Address of the source tensor in global memory.
- **shape** – Pointer to the shape of the source tensor.
- **data_type** – Data type of source tensor.
- **base_idx** – The starting index of destination tensor.

Remarks

- The destination tensor is in compact layout. The source tensor is in compact layout.
- The number of indices can be obtained by `tpu_gdma_get_filter_num()`. `tpu_gdma_get_filter_num()` has no input parameters and its return data type is `DT_UINT32`.

4.6.19 tpu_gdma_compact_S2L

Copy tensor from system memory to local memory.

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

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

Parameters

- **dst_addr** – Address of the destination tensor in local memory.
- **src_addr** – Address of the source tensor in system memory.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dtype** – Data type of source and destination tensor.

Remarks

- The destination tensor is in compact layout. The source tensor is in continuous layout.

4.6.20 tpu_gdma_compact_L2S

Copy tensor from local memory to system memory.

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

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

Parameters

- **dst_addr** – Address of the destination tensor in system memory.
- **src_addr** – Address of the source tensor in local memory.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dtype** – Data type of source and destination tensor.

Remarks

- The destination tensor is in continuous layout. The source tensor is in compact layout.

4.6.21 tpu_gdma_compact_nc_trans_S2L

Copy tensor from system memory to local memory, with N and C dimensions transposed.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor in local memory.
- **src_addr** – Address of the source tensor in system memory.
- **dst_shape** – Pointer to the shape of the destination tensor.
- **dtype** – Data type of source and destination tensor.

Remarks

- The shape of the source tensor is [dst_shape->c, dst_shape->n, dst_shape->h, dst_shape->w].
- The destination tensor is in compact layout. The source tensor is in continuous layout.

4.6.22 tpu_gdma_compact_nc_trans_L2S

Copy tensor from local memory to system memory, with N and C dimensions transposed.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor in system memory.
- **src_addr** – Address of the source tensor in local memory.
- **dst_shape** – Pointer to the shape of the destination tensor.
- **dtype** – Data type of source and destination tensor.

Remarks

- The shape of the source tensor is [dst_shape->c, dst_shape->n, dst_shape->h, dst_shape->w].
- The destination tensor is in continuous layout. The source tensor is in compact layout.

4.6.23 tpu_gdma_set_C_system

Set all the elements of the destination tensor in system memory to a constant value.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor in system memory.
- **C** – Constant value.
- **shape** – Pointer to the shape of the destination tensor.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **dtype** – Data type of destination tensor and *C*.

Remarks

- If **dst_stride** is NULL, the destination tensor is in continuous layout, otherwise in free layout.
- **dst_stride->w** less than or equal to $128 / \text{tpu_data_type_size}(\text{dtype})$.

4.6.24 tpu_gdma_set_C_local

Set all the elements of the destination tensor in local memory to a constant value.

```
void tpu_gdma_set_C_local(local_addr_t dst_addr, scalar_t C, const dim4_t *shape, const
                        dim4_t *dst_stride, data_type_t dtype)
```

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

Parameters

- **dst_addr** – Address of the destination tensor in local memory.
- **C** – Constant value.
- **shape** – Pointer to the shape of the destination tensor.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **dtype** – Data type of destination tensor and *C*.

Remarks

- If **dst_stride** is NULL, the destination tensor is in 64-byte aligned layout, otherwise in free layout.
- **dst_stride**→w less than or equal to 128 / *tpu_data_type_size(dtype)*.

4.6.25 tpu_gdma_matrix_S2L

Copy matrix from system memory to 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)
```

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

Parameters

- **dst_addr** – Address of the destination matrix in local memory.
- **src_addr** – Address of the source matrix in system memory.
- **rows** – Number of rows in the matrix.
- **cols** – Number of columns in the matrix.
- **cols_per_channel** – The number of columns of the destination matrix in each channel.
- **row_stride** – Pointer to the line stride of the source matrix.
- **dtype** – Data type of source and destination matrix.

Remarks

- The destination matrix is in matrix layout. Each row of the source matrix is in continuous layout, and line break through **row_stride**.

4.6.26 tpu_gdma_matrix_L2S

Copy matrix from local memory to 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)
```

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

Parameters

- **dst_addr** – Address of the destination matrix in system memory.
- **src_addr** – Address of the source matrix in local memory.
- **rows** – Number of rows in the matrix.
- **cols** – Number of columns in the matrix.
- **cols_per_channel** – The number of columns of the source matrix in each channel.
- **row_stride** – Pointer to the line stride of the destination matrix.
- **dtype** – Data type of source and destination matrix.

Remarks

- Each row of the destination matrix is in continuous layout, and line break through **row_stride**. The source matrix is in matrix layout.

4.6.27 tpu_gdma_matrix_trans_S2L

Copy matrix from system memory to local memory with transposition.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination matrix in local memory.
- **src_addr** – Address of the source matrix in system memory.
- **src_rows** – Number of rows in the source matrix.
- **src_cols** – Number of columns in the source matrix.
- **dst_cols_per_channel** – The number of columns of the destination matrix in each channel.
- **src_row_stride** – Pointer to the line stride of the source matrix.
- **dtype** – Data type of source and destination matrix.

Remarks

- The destination matrix is in matrix layout. Each row of the source matrix is in continuous layout, and line break through **src_row_stride**.

4.6.28 tpu_gdma_matrix_trans_L2S

Copy matrix from local memory to system memory with transposition.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination matrix in system memory.
- **src_addr** – Address of the source matrix in local memory.
- **src_rows** – Number of rows in the source matrix.
- **src_cols** – Number of columns in the source matrix.
- **src_cols_per_channel** – The number of columns of the source matrix in each channel.
- **dst_row_stride** – Pointer to the line stride of the destination matrix.
- **dtype** – Data type of source and destination matrix.

Remarks

- Each row of the destination matrix is in continuous layout, and line break through **dst_row_stride**. The source matrix is in matrix layout.

4.6.29 tpu_gdma_vector_S2L

Copy vector from system memory to local memory with transposition.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination vector in local memory.
- **src_addr** – Address of the source vector in system memory.
- **len** – The length of the vector.
- **len_per_channel** – The number of length of the destination vector in each channel.
- **dtype** – Data type of source and destination vector.

Remarks

- The destination vector is in vector layout. The source vector is in continuous layout.

4.6.30 tpu_gdma_vector_L2S

Copy vector from local memory to 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)
```

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

Parameters

- **dst_addr** – Address of the destination vector in system memory.
- **src_addr** – Address of the source vector in local memory.
- **len** – The length of the vector.
- **len_per_channel** – The number of length of the source vector in each channel.
- **dtype** – Data type of source and destination vector.

Remarks

- The destination vector is in continuous layout. The source vector is in vector layout.

4.6.31 tpu_gdma_channel_bcast_S2L

Copy tensor from system memory to local memory, with broadcasting on each channel.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor in local memory.
- **src_addr** – Address of the source tensor in system memory.
- **shape** – Pointer to the shape of the destination tensor.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dtype** – Data type of source and destination tensor.

Remarks

- The shape of the source tensor is [**shape->n**, 1, **shape->h**, **shape->w**].
- If **dst_stride** is NULL, the destination tensor is in 64-byte aligned layout, otherwise in free layout.
- If **src_stride** is NULL, the source tensor is in continuous layout, otherwise in free layout.
- If the destination tensor starts at NPU X, X is in [0, **NPU_NUM** - 1], and **shape->c** less than or equal to **NPU_NUM** - X.
- **dst_stride->w** and **src_stride->w** are only allowed to be 1.

4.6.32 tpu_gdma_channel_bcast_L2L

Copy tensor from local memory to local memory, with broadcasting on each channel.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor in local memory.
- **src_addr** – Address of the source tensor in local memory.
- **shape** – Pointer to the shape of the destination tensor.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dtype** – Data type of source and destination tensor.

Remarks

- The shape of the source tensor is [shape->n, 1, shape->h, shape->w].
- If **dst_stride** is NULL, the destination tensor is in 64-byte aligned layout, otherwise in free layout.
- If **src_stride** is NULL, the source tensor is in 64-byte aligned layout, otherwise in free layout.
- If the destination tensor starts at NPU X, X is in the range [0, *NPU_NUM* - 1], and shape->c less than or equal to *NPU_NUM* - X.
- **dst_stride->w** and **src_stride->w** are only allowed to be 1.

4.6.33 tpu_gdma_h_gather_S2L

Gather elements from system memory to local memory by *h* index, that is, 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)
```

$$\text{output}(0, c, h, w) = \begin{cases} \text{param}(0, c, \text{index}(0, c, h, 0), w) & \text{If } \text{index}(0, c, h, 0) \text{ is valid,} \\ C & \text{Otherwise.} \end{cases}$$

Parameters

- **output_addr** – Address of the output tensor in local memory.
- **param_addr** – Address of the param tensor in system memory.
- **index_addr** – Address of the index tensor in system memory or local memory.
- **index_is_local** – Flag to monitor if index is in local memory.
- **C** – Constant value.
- **shape** – Pointer to the shape of the output tensor.

- **param_h** – h of the param tensor.
- **output_stride** – Pointer to the stride of the output tensor.
- **param_stride** – Pointer to the stride of the param tensor.
- **index_stride** – Pointer to the stride of the index tensor.
- **dtype** – Data type of param and output tensor.

Remarks

- If **output_stride** is NULL, the output tensor is in 64-byte aligned layout, otherwise in free layout.
- If **param_stride** is NULL, the param tensor is in continuous layout, otherwise in free layout.
- If **index_stride** is NULL, the index tensor is in 64-byte aligned layout (while **index_is_local** is true) or continuous layout (while **index_is_local** is false), otherwise in free layout.
- Better performance if **index_addr** is divisible by 512.
- **shape->n** are only allowed to be 1. The shape of param tensor is [1, **shape->c**, **param_h**, **shape->w**]. The shape of index tensor is [1, **shape->c**, **shape->h**, 1].
- **output_stride->w**, **param_stride->w** and **index_stride->h** are only allowed to be 1.
- The data type of index tensor is *DT_UINT32*, with the valid value range of [0, **param_h** - 1].

4.6.34 tpu_gdma_h_gather_L2S

Gather elements from local memory to system memory by h index, that is, $\text{output} = \text{param}[\text{index}]$.

```
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_t *shape, int param_h, const dim4_t *output_stride,
                           const dim4_t *param_stride, const dim4_t *index_stride,
                           data_type_t dtype)
```

$$\text{output}(0, c, h, w) = \begin{cases} \text{param}(0, c, \text{index}(0, c, h, 0), w) & \text{If } \text{index}(0, c, h, 0) \text{ is valid,} \\ C & \text{Otherwise.} \end{cases}$$

Parameters

- **output_addr** – Address of the output tensor in system memory.
- **param_addr** – Address of the param tensor in local memory.
- **index_addr** – Address of the index tensor in system memory or local memory.
- **index_is_local** – Flag to monitor if index is in local memory.
- **C** – Constant value.
- **shape** – Pointer to the shape of the output tensor.
- **param_h** – h of the param tensor.
- **output_stride** – Pointer to the stride of the output tensor.
- **param_stride** – Pointer to the stride of the param tensor.
- **index_stride** – Pointer to the stride of the index tensor.
- **dtype** – Data type of param and output tensor.

Remarks

- If `output_stride` is NULL, the output tensor is in continuous layout, otherwise in free layout.
- If `param_stride` is NULL, the param tensor is in 64-byte aligned layout, otherwise in free layout.
- If `index_stride` is NULL, the index tensor is in 64-byte aligned layout (`index_is_local` is true) or continuous layout (`index_is_local` is false), otherwise in free layout.
- Better performance if `index_addr` is divisible by 512.
- `shape->n` are only allowed to be 1. The shape of param tensor is `[1, shape->c, param_h, shape->w]`. The shape of index tensor is `[1, shape->c, shape->h, 1]`.
- `output_stride->w`, `param_stride->w` and `index_stride->h` are only allowed to be 1.
- The data type of index tensor is `DT_UINT32`, with the valid value range of `[0, param_h - 1]`.

4.6.35 tpu_gdma_h_gather_L2L

Gather elements from local memory to local memory by h index, that is, $\text{output} = \text{param}[\text{index}]$.

```
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_t *shape, int param_h, const dim4_t *output_stride,
                           const dim4_t *param_stride, const dim4_t *index_stride,
                           data_type_t dtype)
```

$$\text{output}(0, c, h, w) = \begin{cases} \text{param}(0, c, \text{index}(0, c, h, 0), w) & \text{If } \text{index}(0, c, h, 0) \text{ is valid,} \\ C & \text{Otherwise.} \end{cases}$$

Parameters

- **output_addr** – Address of the output tensor in local memory.
- **param_addr** – Address of the param tensor in local memory.
- **index_addr** – Address of the index tensor in system memory or local memory.
- **index_is_local** – Flag to monitor if index is in local memory.
- **C** – Constant value.
- **shape** – Pointer to the shape of the output tensor.
- **param_h** – h of the param tensor.
- **output_stride** – Pointer to the stride of the output tensor.
- **param_stride** – Pointer to the stride of the param tensor.
- **index_stride** – Pointer to the stride of the index tensor.
- **dtype** – Data type of param and output tensor.

Remarks

- If `output_stride` is NULL, the output tensor is in 64-byte aligned layout, otherwise in free layout.
- If `param_stride` is NULL, the param tensor is in 64-byte aligned layout, otherwise in free layout.

- If `index_stride` is NULL, the index tensor is in 64-byte aligned layout (`index_is_local` is true) or continuous layout (`index_is_local` is false), otherwise in free layout.
- Better performance if `index_addr` is divisible by 512.
- `shape->n` are only allowed to be 1. The shape of param tensor is [1, `shape->c`, `param_h`, `shape->w`]. The shape of index tensor is [1, `shape->c`, `shape->h`, 1].
- `output_stride->w`, `param_stride->w` and `index_stride->h` are only allowed to be 1.
- The data type of index tensor is `DT_UINT32`, with the valid value range of [0, `param_h` - 1].

4.6.36 tpu_gdma_h_gather_S2S

Gather elements from system memory to system memory by `h` index, that is, `output = param[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)
```

$$\text{output}(0, c, h, w) = \begin{cases} \text{param}(0, c, \text{index}(0, c, h, 0), w) & \text{If } \text{index}(0, c, h, 0) \text{ is valid,} \\ C & \text{Otherwise.} \end{cases}$$

Parameters

- `output_addr` – Address of the output tensor in system memory.
- `param_addr` – Address of the param tensor in system memory.
- `index_addr` – Address of the index tensor in system memory or local memory.
- `index_is_local` – Flag to monitor if index is in local memory.
- `C` – Constant value.
- `shape` – Pointer to the shape of the output tensor.
- `param_h` – `h` of the param tensor.
- `output_stride` – Pointer to the stride of the output tensor.
- `param_stride` – Pointer to the stride of the param tensor.
- `index_stride` – Pointer to the stride of the index tensor.
- `dtype` – Data type of param and output tensor.

Remarks

- If `output_stride` is NULL, the output tensor is in continuous layout, otherwise in free layout.
- If `param_stride` is NULL, the param tensor is in continuous layout, otherwise in free layout.
- If `index_stride` is NULL, the index tensor is in 64-byte aligned layout (`index_is_local` is true) or continuous layout (`index_is_local` is false), otherwise in free layout.
- Better performance if `index_addr` is divisible by 512.
- `shape->n` are only allowed to be 1. The shape of param tensor is [1, `shape->c`, `param_h`, `shape->w`]. The shape of index tensor is [1, `shape->c`, `shape->h`, 1].
- `output_stride->w`, `param_stride->w` and `index_stride->h` are only allowed to be 1.

- The data type of index tensor is *DT_UINT32*, with the valid value range of $[0, \text{param_h} - 1]$.

4.6.37 tpu_gdma_h_scatter_S2L

Scatter elements from system memory to local memory by *h* index, that is, $\text{output}[\text{index}] = \text{param}$.

```
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)
```

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

Parameters

- **output_addr** – Address of the output tensor in local memory.
- **param_addr** – Address of the param tensor in system memory.
- **index_addr** – Address of the index tensor in system memory or local memory.
- **index_is_local** – Flag to monitor if index is in local memory.
- **shape** – Pointer to the shape of the output tensor.
- **param_h** – *h* of the param tensor.
- **output_stride** – Pointer to the stride of the output tensor.
- **param_stride** – Pointer to the stride of the param tensor.
- **index_stride** – Pointer to the stride of the index tensor.
- **dtype** – Data type of param and output tensor.

Remarks

- If **output_stride** is NULL, the output tensor is in 64-byte aligned layout, otherwise in free layout.
- If **param_stride** is NULL, the param tensor is in continuous layout, otherwise in free layout.
- If **index_stride** is NULL, the index tensor is in 64-byte aligned layout (**index_is_local** is true) or continuous layout (**index_is_local** is false), otherwise in free layout.
- Better performance if **index_addr** is divisible by 512.
- **shape->n** are only allowed to be 1. The shape of param tensor is $[1, \text{shape->c}, \text{param_h}, \text{shape->w}]$. The shape of index tensor is $[1, \text{shape->c}, \text{param_h}, 1]$.
- **output_stride->w**, **param_stride->w** and **index_stride->h** are only allowed to be 1.
- The data type of index tensor is *DT_UINT32*, with the valid value range of $[0, \text{shape->n} - 1]$.

4.6.38 tpu_gdma_h_scatter_L2S

Scatter elements from local memory to system memory by *h* index, that is, `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)
```

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

Parameters

- **output_addr** – Address of the output tensor in system memory.
- **param_addr** – Address of the param tensor in local memory.
- **index_addr** – Address of the index tensor in system memory or local memory.
- **index_is_local** – Flag to monitor if index is in local memory.
- **shape** – Pointer to the shape of the output tensor.
- **param_h** – *h* of the param tensor.
- **output_stride** – Pointer to the stride of the output tensor.
- **param_stride** – Pointer to the stride of the param tensor.
- **index_stride** – Pointer to the stride of the index tensor.
- **dtype** – Data type of param and output tensor.

Remarks

- If **output_stride** is NULL, the output tensor is in continuous layout, otherwise in free layout.
- If **param_stride** is NULL, the param tensor is in 64-byte aligned layout, otherwise in free layout.
- If **index_stride** is NULL, the index tensor is in 64-byte aligned layout (**index_is_local** is true) or continuous layout (**index_is_local** is false), otherwise in free layout.
- Better performance if **index_addr** is divisible by 512.
- **shape->n** are only allowed to be 1. The shape of param tensor is [1, **shape->c**, **param_h**, **shape->w**]. The shape of index tensor is [1, **shape->c**, **param_h**, 1].
- **output_stride->w**, **param_stride->w** and **index_stride->h** are only allowed to be 1.
- The data type of index tensor is *DT_UINT32*, with the valid value range of [0, **shape->h** - 1].

4.6.39 tpu_gdma_h_scatter_L2L

Scatter elements from local memory to local memory by h index, that is, $\text{output}[\text{index}] = \text{param}$.

```
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)
```

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

Parameters

- **output_addr** – Address of the output tensor in local memory.
- **param_addr** – Address of the param tensor in local memory.
- **index_addr** – Address of the index tensor in system memory or local memory.
- **index_is_local** – Flag to monitor if index is in local memory.
- **shape** – Pointer to the shape of the output tensor.
- **param_h** – h of the param tensor.
- **output_stride** – Pointer to the stride of the output tensor.
- **param_stride** – Pointer to the stride of the param tensor.
- **index_stride** – Pointer to the stride of the index tensor.
- **dtype** – Data type of param and output tensor.

Remarks

- If **output_stride** is NULL, the output tensor is in 64-byte aligned layout, otherwise in free layout.
- If **param_stride** is NULL, the param tensor is in 64-byte aligned layout, otherwise in free layout.
- If **index_stride** is NULL, the index tensor is in 64-byte aligned layout (**index_is_local** is true) or continuous layout (**index_is_local** is false), otherwise in free layout.
- Better performance if **index_addr** is divisible by 512.
- **shape->n** are only allowed to be 1. The shape of param tensor is [1, **shape->c**, **param_h**, **shape->w**]. The shape of index tensor is [1, **shape->c**, **param_h**, 1].
- **output_stride->w**, **param_stride->w** and **index_stride->h** are only allowed to be 1.
- The data type of index tensor is *DT_UINT32*, with the valid value range of [0, **shape->h** - 1].

4.6.40 tpu_gdma_h_scatter_S2S

Scatter elements from system memory to system memory by h index, that is, $\text{output}[\text{index}] = \text{param}$.

```
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)
```

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

Parameters

- **output_addr** – Address of the output tensor in system memory.
- **param_addr** – Address of the param tensor in system memory.
- **index_addr** – Address of the index tensor in system memory or local memory.
- **index_is_local** – Flag to monitor if index is in local memory.
- **shape** – Pointer to the shape of the output tensor.
- **param_h** – h of the param tensor.
- **output_stride** – Pointer to the stride of the output tensor.
- **param_stride** – Pointer to the stride of the param tensor.
- **index_stride** – Pointer to the stride of the index tensor.
- **dtype** – Data type of param and output tensor.

Remarks

- If **output_stride** is NULL, the output tensor is in continuous layout, otherwise in free layout.
- If **param_stride** is NULL, the param tensor is in continuous layout, otherwise in free layout.
- If **index_stride** is NULL, the index tensor is in 64-byte aligned layout (**index_is_local** is true) or continuous layout (**index_is_local** is false), otherwise in free layout.
- Better performance if **index_addr** is divisible by 512.
- **shape->n** are only allowed to be 1. The shape of param tensor is [1, **shape->c**, **param_h**, **shape->w**]. The shape of index tensor is [1, **shape->c**, **param_h**, 1].
- **output_stride->w**, **param_stride->w** and **index_stride->h** are only allowed to be 1.
- The data type of index tensor is *DT_UINT32*, with the valid value range of [0, **shape->h** - 1].

4.6.41 tpu_gdma_system_cpy

Copy data from system memory to system memory.

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

Parameters

- **dst_addr** – Address of the destination in system memory.
- **src_addr** – Address of the source in system memory.
- **count** – Length of data.
- **dtype** – Data type of source and destination.

Remarks

- **dst_addr** and **src_addr** are both divisible by the bit width of the data type.

4.6.42 tpu_gdma_reverse_S2S

Reverse tensor data order by the specified axis (N or C or H) from one System Memory to the other

```
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);
```

Parameters

- **dst_addr** – Address of the destination in system memory.
- **src_addr** – Address of the source in system memory.
- **shape** – Pointer to the shape of the tensor.
- **dst_stride** – Pointer to the destination tensor stride
- **src_stride** – Pointer to the source tensor stride
- **reverse_axis** – The specified axis, support 0-N, 1-C, 2-H
- **dtype** – Data type of the tensor

Remarks

- NOT Support BM1684X
- *dst_stride* and *src_stride* can be NULL when layout is continuous
- The provided valid *src_stride* must satisfy ‘src_stride->w’==1
- The provided valid *dst_stride* must satisfy ‘src_stride->w’==1

4.6.43 tpu_gdma_reverse_S2L

Reverse tensor data order by the specified axis (N or C or H) from the System Memory to the Local Memory

```
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);
```

Parameters

- **dst_addr** – Address of the destination in local memory.
- **src_addr** – Address of the source in system memory.
- **shape** – Pointer to the shape of the tensor.
- **dst_stride** – Pointer to the destination tensor stride
- **src_stride** – Pointer to the source tensor stride
- **reverse_axis** – The specified axis, support 0-N, 1-C, 2-H
- **dtype** – Data type of the tensor

Remarks

- NOT Support BM1684X
- *dst_stride* and *src_stride* can be NULL when layout is continuous
- The provided valid *src_stride* must satisfy ‘src_stride->w’==1
- The provided valid *dst_stride* must satisfy ‘src_stride->w’==1

4.6.44 tpu_gdma_reverse_L2S

Reverse tensor data order by the specified axis (C only) from the Local Memory to the System Memory

```
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);
```

Parameters

- **dst_addr** – Address of the destination in system memory.
- **src_addr** – Address of the source in local memory.
- **shape** – Pointer to the shape of the tensor.
- **dst_stride** – Pointer to the destination tensor stride
- **src_stride** – Pointer to the source tensor stride
- **reverse_axis** – The specified axis, support 1-C
- **dtype** – Data type of the tensor

Remarks

- NOT Support BM1684X
- *dst_stride* and *src_stride* can be NULL when layout is continuous
- The provided valid *src_stride* must satisfy ‘src_stride->w’==1
- The provided valid *dst_stride* must satisfy ‘src_stride->w’==1

4.6.45 tpu_gdma_reverse_L2L

Reverse tensor data order by the specified axis (C only) from one Local Memory to the other one

```
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);
```

Parameters

- **dst_addr** – Address of the destination in local memory.
- **src_addr** – Address of the source in local memory.
- **shape** – Pointer to the shape of the tensor.
- **dst_stride** – Pointer to the destination tensor stride
- **src_stride** – Pointer to the source tensor stride
- **reverse_axis** – The specified axis, support 1-C
- **dtype** – Data type of the tensor

Remark

- NOT Support BM1684X
- *dst_stride* and *src_stride* can be NULL when layout is continuous
- The provided valid *src_stride* must satisfy ‘src_stride->w’==1
- The provided valid *dst_stride* must satisfy ‘src_stride->w’==1

4.6.46 tpu_gdma_compress_normal_L2S

Compress data in Local Memory to the System Memory. The compressed data cannot be random accessed, so it’s used to load or store the whole data once

```
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);
```

Parameters

- **dst_addr** – Address of the destination in system memory.
- **src_addr** – Address of the source in local memory.
- **shape** – Pointer to the shape of the tensor.
- **src_stride** – Pointer to the source tensor stride
- **dtype** – Data type of the tensor
- **bias0** – One of compression parameter
- **bias1** – One of compression parameter

Remarks

- NOT Support BM1684X
- *src_stride* can be NULL when layout is continuous
- The provided valid *src_stride* must satisfy ‘src_stride->w’==1
- Compression parameters(bias0、 bias1 8-bit integer) can control the compressing ratio, but need to try

4.6.47 tpu_gdma_decompress_normal_S2L

Decompress the Data in the System Memory compressed by *tpu_gdma_compress_normal_L2S* to 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);
```

Parameters

- **dst_addr** – Address of the destination in system memory.
- **src_addr** – Address of the source in local memory.
- **shape** – Pointer to the shape of the tensor.
- **dst_stride** – Pointer to the destination tensor stride
- **dtype** – Data type of the tensor

Remarks

- NOT Support BM1684X
- *src_stride* can be NULL when layout is continuous
- The provided valid *src_stride* must satisfy '*src_stride->w*'==1

tpu_gdma_compress_normal_max_bytes

Calculate the max byte number of the normal compressed data, to allocate system memory ahead.

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

Parameters

- **shape** – Pointer to the uncompressed data shape
- **dtype** – Data type of the uncompressed data

4.6.48 tpu_gdma_compress_RACU_L2S

Compress the tensor data in Local Memory to the System Memory. And support to be random accessed in size of RACU (Random Access Compression Unit). The compressed data consists of two parts (the RACU and Meta data).

```
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);
```

Parameters

- **dst_racu_addr** – Address of the destination racu data in system memory.
- **dst_meta_addr** – Address of the destination meta data in system memory.
- **src_addr** – Address of the source in local memory.
- **shape** – Pointer to the shape of the tensor.
- **dst_racu_stride** – Output, racu data stride will be calculated after data is compressed, must be used for decompressing
- **dst_meta_stride** – Output, meta data stride will be calculated after data is compressed, must be used for decompressing

- **src_stride** – Pointer to the source tensor stride
- **dtype** – Data type of the tensor
- **bias0** – One of compression parameter
- **bias1** – One of compression parameter

Remarks

- NOT Support BM1684X
- *src_stride* can be NULL when layout is continuous
- The provided valid *src_stride* must satisfy '*src_stride->w*'==1
- Compression parameters(bias0、bias1 8-bit integer) can control the compressing ratio, but need to try

4.6.49 tpu_gdma_decompress_RACU_S2L

Decompress the tensor data in System Memory compressed by *tpu_gdma_compress_RACU_L2S* to Local Memory

```
void tpu_gdma_decompress_RACU_S2L(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)
```

Parameters

- **dst_addr** – Address of the destination in local memory.
- **src_racu_addr** – Address of the source racu data in system memory.
- **src_meta_addr** – Address of the source meta data in system memory.
- **shape** – Pointer to the shape of the tensor.
- **dst_stride** – Pointer to the destination tensor stride
- **src_racu_stride** – Pointer to the stride of the source racu data. Only n, c is used
- **src_meta_stride** – Pointer to the stride of the source meta data. Only n, c is used
- **dtype** – Data type of the tensor
- **bias0** – One of compression parameter, must keep same with the value used in compressing process
- **bias1** – One of compression parameter, must keep same with the value used in compressing process

Remarks

- NOT Support BM1684X
- *src_stride* can be NULL when layout is continuous
- The provided valid *src_stride* must satisfy '*src_stride->w*'==1

tpu_gdma_decompress_RACU_max_racu_bytes

Calculate the max byte number of the generated racu data, to allocate system memory ahead.

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

Parameters

- **shape** – Pointer to the uncompressed data shape
- **dtype** – Data type of the uncompressed data

tpu_gdma_compress_RACU_max_meta_bytes

Calculate the max byte number of the generated meta data, to allocate system memory ahead.

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

Parameters

- **shape** – Pointer to the uncompressed data shape
- **dtype** – Data type of the uncompressed data

4.7 Basic BDC Functions

4.7.1 tpu_bdc_cpy

Copy the elements of the source tensor to the destination tensor.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dtype** – Data type of source and destination tensor.

Remarks

- The destination and source tensors start at the same NPU.
- shape->n, shape->c, shape->h and shape->w are in [1, 65535].
- If dst_stride or src_stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.

4.7.2 tpu_bdc_cpy_cross_npu

Copy the tensor cross npu.

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

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dtype** – Data type of source and destination tensor.

Remarks

- The destination and source tensors may not start at the same NPU, and both are in 64-byte aligned layout.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535].

4.7.3 tpu_bdc_npu_bcast

Broadcast the tensor to other NPU.

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

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **shape** – Pointer to the shape of the destination tensor.
- **dtype** – Data type of source and destination tensor.

Remarks

- The destination and source tensors may not start at the same NPU, and both are in 64-byte aligned layout.
- The shape of the source tensor is [**shape->n**, 1, **shape->h**, **shape->w**].
- **shape->n**, **shape->h** and **shape->w** are in [1, 65535].
- If the destination tensor starts at NPU X, X is in [0, *NPU_NUM* - 1], and **shape->c** less than or equal to *NPU_NUM* - X.

4.7.4 tpu_bdc_set_C

Set all the elements of the destination tensor in local memory to a constant value.

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

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **C** – Constant value.
- **shape** – Pointer to the shape of the destination tensor.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **dtype** – Data type of destination tensor and **C**.

Remarks

- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If `dst_stride` is NULL, the tensor is in 64-byte aligned layout, otherwise in free layout.

4.7.5 tpu_bdc_cw_trans

Transposes the dimensions C and W of the tensor. It is recommended to use when $C > W$.

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

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **shape** – Pointer to the shape of the destination tensor.
- **dtype** – Data type of source and destination tensor.

Remarks

- The source tensor starts at NPU 0, and is in 64-byte aligned layout. The destination tensor is in line 64-byte aligned layout.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].

4.7.6 tpu_bdc_wc_trans

Transposes the dimensions W and C of the tensor. It is recommended to use when $W > C$.

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

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **shape** – Pointer to the shape of the destination tensor.
- **dtype** – Data type of source and destination tensor.

Remarks

- The destination tensor starts at NPU 0. The source and destination tensors are both in 64-byte aligned layout.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].

4.8 Data Conversion and Rounding Functions

4.8.1 tpu_bdc_cast

Convert the data type of the source tensor and perform saturation on the result.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dst_dtype** – Data type of destination tensor.
- **src_dtype** – Data type of source tensor.
- **mode** – Rounding mode.

Remarks

- The destination and source tensors start at the same NPU.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535].
- If the stride is NULL, the tensor is in 64-byte aligned layout, otherwise in free layout.
- It does not support the data type conversion between *DT_FP16* and *DT_BFP16*. If **dst_dtype** is the same as **src_dtype** the elements of destination and source tensors are exactly the same.
- The **mode** is only used when the floating-point data is involved, the valid values are: *RM_HALF_TO_EVEN*, *RM_HALF_AWAY_FROM_ZERO*, *RM_TOWARDS_ZERO*, *RM_DOWN* and *RM_UP*.
- The following table lists whether **mode** is valid. The ★ means valid, and blank means invalid.

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

Round the source tensor of floating-point type to the nearest integer.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dtype** – Data type of source and destination tensor.
- **mode** – Rounding mode.

Remarks

- The destination and source tensors start at the same NPU.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535].
- If the stride is NULL, the tensor is in 64-byte aligned layout, otherwise in free layout.
- The valid choices of **dtype** are *DT_FP32*, *DT_FP16* and *DT_BFP16*.
- The valid choices of **mode** are *RM_HALF_TO_EVEN*, *RM_HALF_AWAY_FROM_ZERO*, *RM_TOWARDS_ZERO*, *RM_DOWN* and *RM_UP*.

4.8.3 tpu_bdc_fp_floor

Round the source tensor of floating-point type to the nearest integer toward negative infinity.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dtype** – Data type of source and destination tensor.

Remarks

- The destination and source tensors start at the same NPU.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535].

- If the stride is NULL, the tensor is in 64-byte aligned layout, otherwise in free layout.
- The valid choices of `dtype` are `DT_FP32`, `DT_FP16` and `DT_BFP16`.

4.8.4 tpu_bdc_fp_ceil

Round the source tensor of floating-point type to the nearest integer toward positive infinity.

```
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)
```

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

Parameters

- `dst_addr` – Address of the destination tensor.
- `src_addr` – Address of the source tensor
- `shape` – Pointer to the shape of the destination and source tensors.
- `dst_stride` – Pointer to the stride of the destination tensor.
- `src_stride` – Pointer to the stride of the source tensor.
- `dtype` – Data type of source and destination tensor.

Remarks

- The destination and source tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If the stride is NULL, the tensor is in 64-byte aligned layout, otherwise in free layout.
- The valid choices of `dtype` are `DT_FP32`, `DT_FP16` and `DT_BFP16`.

4.9 Unary Functions

4.9.1 tpu_bdc_abs

Get the absolute value of the tensor.

```
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)
```

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

Parameters

- `dst_addr` – Address of the destination tensor.
- `src_addr` – Address of the source tensor.
- `shape` – Pointer to the shape of the destination and source tensors.
- `dst_stride` – Pointer to the stride of the destination tensor.
- `src_stride` – Pointer to the stride of the source tensor.
- `dtype` – Data type of source and destination tensor.

Remarks

- The destination and source tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If `dst_stride` or `src_stride` is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.

4.9.2 tpu_bdc_not

Reverse the elements of the source tensor in binary form.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dtype** – Data type of source and destination tensor.

Remarks

- The destination and source tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If `dst_stride` or `src_stride` is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.

4.9.3 tpu_bdc_neg

Calculate negative of the source tensor.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dtype** – Data type of source and destination tensor.

Remarks

- The destination and source tensors start at the same NPU.

- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If `dst_stride` or `src_stride` is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- If `dtype` is `DT_UINT32`, `DT_UINT16` or `DT_UINT8`, the data type of destination tensor will be `DT_INT32`, `DT_INT16` or `DT_INT8`.

4.9.4 tpu_bdc_fp32_reciprocal

Calculate the reciprocal of the source tensor.

```
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)
```

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

Parameters

- `dst_addr` – Address of the destination tensor.
- `src_addr` – Address of the source tensor.
- `shape` – Pointer to the shape of the destination and source tensors.
- `dst_stride` – Pointer to the stride of the destination tensor.
- `src_stride` – Pointer to the stride of the source tensor.

Remarks

- The destination and source tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If `dst_stride` or `src_stride` is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- It is equivalent to `tpu_bdc_fp32_tunable_reciprocal()` with a Newton iteration of 3.

4.9.5 tpu_bdc_fp32_tunable_reciprocal

Calculate the reciprocal of the source tensor.

```
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)
```

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

Parameters

- `dst_addr` – Address of the destination tensor.
- `src_addr` – Address of the source tensor.
- `shape` – Pointer to the shape of the destination and source tensors.
- `dst_stride` – Pointer to the stride of the destination tensor.
- `src_stride` – Pointer to the stride of the source tensor.
- `num_iter` – Newton iterations.

Remarks

- The destination and source tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If `dst_stride` or `src_stride` is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- `num_iter` is a performance parameter, with a range of [1, 4].

4.9.6 tpu_bdc_fp32_rsqr

Calculate reciprocal of the square-root of the source tensor.

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

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **shape** – Pointer to the shape of the destination and source tensors.

Remarks

- The destination and source tensors start at the same NPU, and both are in 64-byte aligned layout.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- It is equivalent to `tpu_bdc_fp32_tunable_rsqr()` with a Newton iteration of 3.

4.9.7 tpu_bdc_fp32_tunable_rsqr

Calculate reciprocal of the square-root of the source tensor.

```
void tpu_bdc_fp32_tunable_rsqr(local_addr_t dst_addr, local_addr_t src_addr, const
                               dim4 *shape, int num_iter)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **shape** – Pointer to the shape of the destination and source tensors.
- **num_iter** – Newton iterations.

Remarks

- The destination and source tensors start at the same NPU, and both are in 64-byte aligned layout.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- `num_iter` is a performance parameter, with a range of [1, 4].

4.9.8 tpu_bdc_fp32_sqrt

Calculate square-root of the source tensor.

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

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **shape** – Pointer to the shape of the destination and source tensors.

Remarks

- The destination and source tensors start at the same NPU, and both are in 64-byte aligned layout. **dst_addr** and **src_addr** must be different.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535].
- It is equivalent to `tpu_bdc_fp32_tunable_sqrt()` with a Newton iteration of 3.

4.9.9 tpu_bdc_fp32_tunable_sqrt

Calculate square-root of the source tensor.

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

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **shape** – Pointer to the shape of the destination and source tensors.
- **num_iter** – Newton iterations.

Remarks

- The destination and source tensors start at the same NPU, and both are in 64-byte aligned layout. **dst_addr** and **src_addr** must be different.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535].
- **num_iter** is a performance parameter, with a range of [1, 4].

4.9.10 tpu_bdc_fp32_exp

Calculate exponential of the source tensor.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **work0_addr** – Address of the work0 tensor.
- **work1_addr** – Address of the work1 tensor.
- **coeff_addr** – Address of the coeff.
- **table_addr** – Address of the table.
- **shape** – Pointer to the shape of the destination, source, work0 and work1 tensors.

Remarks

- The destination, source, work0 and work1 tensors start at the same NPU, and all are in 64-byte aligned layout.
- Coeff is loaded by `tpu_bdc_load_fp32_exp_coeff()`, table is loaded by `tpu_bdc_load_fp32_exp_table()`.
- The work0 and work1 tensors are workspaces to store temporary tensors. Any two of **dst_addr**, **work0_addr** and **work1_addr** are not allowed to be equal. **src_addr** is equal to: `cpp:expr:dst_addr`, `work0_addr` or `work1_addr`, in this case, source data will be overwritten. If there is no bank conflicting between any two of destination, work1 and table tensors, the performance will be better.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in $[1, 65535]$, **shape->h * shape->w** less than or equal to 65535.
- If **src(n, c, h, w)** is less than -103 or greater than 88, then $e^{\text{src}(n,c,h,w)}$ will be e^{-103} or e^{88} , respectively.

4.9.11 tpu_bdc_fp32_expm1

Calculate exponential of the source tensor, and the result is subtracted by 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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **work0_addr** – Address of the work0 tensor.
- **work1_addr** – Address of the work1 tensor.
- **coeff_addr** – Address of the coeff tensor.
- **table_addr** – Address of the table tensor.
- **shape** – Pointer to the shape of the destination, source, work0 and work1 tensors.

Remarks

- The destination, source, work0 and work1 tensors start at the same NPU, and all are in 64-byte aligned layout.

- Coeff is loaded by `tpu_bdc_load_fp32_exp_coeff()`, table is loaded by `tpu_bdc_load_fp32_exp_table()`.
- The work0 and work1 tensors are workspaces to store temporary tensors. Any two of `dst_addr`, `work0_addr` and `work1_addr` are not allowed to be equal. `src_addr` is equal to: `cpp:expr:dst_addr`, `work0_addr` or `work1_addr`, in this case, source data will be overwritten. If there is no bank conflicting between any two of destination, work1 and table tensors, the performance will be better.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in $[1, 65535]$, `shape->h * shape->w` less than or equal to 65535.
- If `src(n, c, h, w)` is less than -103 or greater than 88, then $e^{\text{src}(n, c, h, w)}$ will be e^{-103} or e^{88} , respectively.

4.9.12 tpu_bdc_fp32_log

Calculate logarithm of the source tensor, with natural constant e as base.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **work_addr** – Address of the work tensor.
- **coeff_addr** – Address of the coeff tensor.
- **shape** – Pointer to the shape of the destination, source and work tensors.

Remarks

- The destination, source and work tensors start at the same NPU, and all are in 64-byte aligned layout.
- Coeff is loaded by `tpu_bdc_load_fp32_log_coeff()`.
- The work tensor is a workspace to store temporary tensor. `dst_addr` is not equal to `work_addr`. `src_addr` is equal to `dst_addr` or `work_addr`. In this case, the data of the source tensor will be overwritten.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in $[1, 65535]$.

4.9.13 tpu_bdc_fp32_log1p

Calculate logarithm of the result of adding 1 to the source tensor, with natural constant e as base.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.

- **work_addr** – Address of the work tensor.
- **coeff_addr** – Address of the coeff tensor.
- **shape** – Pointer to the shape of the destination, source and work tensors.

Remarks

- The destination, source and work tensors start at the same NPU, and all are in 64-byte aligned layout.
- Coeff is loaded by `tpu_bdc_load_fp32_log_coeff()`.
- The work tensor is a workspace to store temporary tensor. **dst_addr** is not equal to **work_addr**. **src_addr** is equal to **dst_addr** or **work_addr**. In this case, the data of the source will be overwritten.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535].

4.9.14 tpu_bdc_fp32_logx

Calculate logarithm of the source tensor, with x as base.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **work_addr** – Address of the work tensor.
- **coeff_addr** – Address of the coeff tensor.
- **shape** – Pointer to the shape of the destination, source and work tensors.
- **x** – Base.

Remarks

- The destination, source and work tensors start at the same NPU, and all are in 64-byte aligned layout.
- Coeff is loaded by `tpu_bdc_load_fp32_log_coeff()`.
- The work tensor is a workspace to store temporary tensor. **dst_addr** is not equal to **work_addr**. **src_addr** is equal to **dst_addr** or **work_addr**. In this case, the data of the source will be overwritten.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535].
- $x > 0$.

4.9.15 tpu_bdc_sign

Calculate *sign* of the source tensor.

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

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **shape** – Pointer to the shape of the destination tensors.
- **dtype** – Data type of source and destination tensor.

Remarks

- The destination and source tensors start at the same NPU, and both are in 64-byte aligned layout.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535].
- If **dtype** is unsigned, allow **dst_addr** to be the same as **src_addr**.

4.9.16 tpu_bdc_fp32_sin

Calculate sine of the elements of the source tensor.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **work_addr** – Address of the work tensor.
- **coeff_addr** – Address of the coeff tensor.
- **shape** – Pointer to the shape of the destination, source and work tensors.

Remarks

- The destination, source and work tensors start at the same NPU, and all are in 64-byte aligned layout.
- Coeff is loaded by `tpu_bdc_load_fp32_sin_coeff()`.
- The work tensor is a workspace to store temporary tensor. **dst_addr** is not equal to **work_addr**. **src_addr** is equal to: `cpp:expr:dst_addr` or `work_addr`. In this case, the data of the source tensor will be overwritten.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535].

4.9.17 tpu_bdc_fp32_cos

Calculate cosine of the elements of the source tensor.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **work_addr** – Address of the work tensor.
- **coeff_addr** – Address of the coeff tensor.
- **shape** – Pointer to the shape of the destination, source and work tensors.

Remarks

- The destination, source and work tensors start at the same NPU, and all are in 64-byte aligned layout.
- Coeff is loaded by `tpu_bdc_load_fp32_cos_coeff()`.
- The work tensor is a workspace to store temporary tensor. **dst_addr** is not equal to **work_addr**. **src_addr** is equal to: `expr:dst_addr` or `work_addr`. In this case, the data of the source tensor will be overwritten.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].

4.9.18 tpu_bdc_fp32_tan

Calculate tangent of the elements of the source tensor.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **work_addr** – Address of the work tensor.
- **coeff_addr** – Address of the coeff tensor.
- **shape** – Pointer to the shape of the destination, source and work tensors.

Remarks

- The destination, source and work tensors start at the same NPU, and all are in 64-byte aligned layout.
- Coeff is loaded by `tpu_bdc_load_fp32_tan_coeff()`.
- The work tensor is a workspace to store temporary tensor. Any two of **dst_addr**, **src_addr** and **work_addr** are not allowed to be equal.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].

4.9.19 tpu_bdc_fp32_cot

Calculate cotangent of the elements of the source tensor.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **work_addr** – Address of the work tensor.
- **coeff_addr** – Address of the coeff tensor.
- **shape** – Pointer to the shape of the destination, source and work tensors.

Remarks

- The destination, source and work tensors start at the same NPU, and all are in 64-byte aligned layout.
- Coeff is loaded by `tpu_bdc_load_fp32_tan_coeff()`.
- The work tensor is a workspace to store temporary tensor. Any two of **dst_addr**, **src_addr** and **work_addr** are not allowed to be equal.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].

4.9.20 tpu_bdc_fp32_arcsin

Calculate arcsine of the elements of the source tensor.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **work_addr** – Address of the work tensor.
- **coeff_addr** – Address of the coeff tensor.
- **shape** – Pointer to the shape of the destination, source and work tensors.

Remarks

- The destination, source and work tensors start at the same NPU, and all are in 64-byte aligned layout.
- Coeff is loaded by `tpu_bdc_load_fp32_arcsin_coeff()`.
- The work tensor is a workspace to store temporary tensor. **dst_addr** is not equal to **work_addr**. **src_addr** is equal to: `cpp:expr:dst_addr` or `work_addr`. In this case, the data of the source tensor will be overwritten.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- Element of source tensor is in $[-1, 1]$. Element of destination tensor is in $[-\frac{\pi}{2}, \frac{\pi}{2}]$.

4.9.21 tpu_bdc_fp32_arccos

Calculate arccosine of the elements of the source tensor.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **work_addr** – Address of the work tensor.
- **coeff_addr** – Address of the coeff tensor.
- **shape** – Pointer to the shape of the destination, source and work tensors.

Remarks

- The destination, source and work tensors start at the same NPU, and all are in 64-byte aligned layout.
- Coeff is loaded by `tpu_bdc_load_fp32_arcsin_coeff()`.
- The work tensor is a workspace to store temporary tensor. **dst_addr** is not equal to **work_addr**. **src_addr** is equal to: `cpp:expr:dst_addr` or `work_addr`. In this case, the data of the source tensor will be overwritten.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535].
- Element of source tensor is in [-1, 1]. Element of destination tensor is in [0, π].

4.10 Binary Functions

4.10.1 tpu_bdc_and

Perform bitwise AND operation of the src0 and src1 tensors.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src0_addr** – Address of the src0 tensor.
- **src1_addr** – Address of the src1 tensor.
- **shape** – Pointer to the shape of the destination, src0 and src1 tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src0_stride** – Pointer to the stride of the src0 tensor.
- **src1_stride** – Pointer to the stride of the src1 tensor.
- **dtype** – Data type of the destination, src0 and src1 tensors.

Remarks

- The destination, src0 and src1 tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535]
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.

4.10.2 tpu_bdc_and_C

Perform bitwise AND operation of the source tensor and a constant value.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **C** – Constant value.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dtype** – Data type of destination, source tensors and *C*.

Remarks

- The destination and source tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.

4.10.3 tpu_bdc_or

Perform bitwise OR operation of the src0 and src1 tensors.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src0_addr** – Address of the src0 tensor.
- **src1_addr** – Address of the src1 tensor.
- **shape** – Pointer to the shape of the destination, src0 and src1 tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src0_stride** – Pointer to the stride of the src0 tensor.

- **src1_stride** – Pointer to the stride of the src1 tensor.
- **dtype** – Data type of the destination, src0 and src1 tensors.

Remarks

- The destination, src0 and src1 tensors start at the same NPU.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.

4.10.4 tpu_bdc_or_C

Perform bitwise OR operation of the source tensor and a constant value.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **C** – Constant value.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dtype** – Data type of destination, source tensors and **C**.

Remarks

- The destination and source tensors start at the same NPU.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.

4.10.5 tpu_bdc_xor

Perform bitwise XOR operation of the src0 and src1 tensors.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src0_addr** – Address of the src0 tensor.
- **src1_addr** – Address of the src1 tensor.
- **shape** – Pointer to the shape of the destination, src0 and src1 tensors.

- **dst_stride** – Pointer to the stride of the destination tensor.
- **src0_stride** – Pointer to the stride of the src0 tensor.
- **src1_stride** – Pointer to the stride of the src1 tensor.
- **dtype** – Data type of the destination, src0 and src1 tensors.

Remarks

- The destination, src0 and src1 tensors start at the same NPU.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.

4.10.6 tpu_bdc_xor_C

Perform bitwise XOR operation of the source tensor and a constant value.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **C** – Constant value.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dtype** – Data type of destination, source tensors and **C**.

Remarks

- The destination and source tensors start at the same NPU.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.

4.10.7 tpu_bdc_min

Perform minimum operation of the src0 and src1 tensors.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src0_addr** – Address of the src0 tensor.

- **src1_addr** – Address of the src1 tensor.
- **shape** – Pointer to the shape of the destination, src0 and src1 tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src0_stride** – Pointer to the stride of the src0 tensor.
- **src1_stride** – Pointer to the stride of the src1 tensor.
- **dtype** – Data type of the destination, src0 and src1 tensors.

Remarks

- The destination, src0 and src1 tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.

4.10.8 tpu_bdc_min_C

Perform minimum operation of the source tensor and a constant value.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **C** – Constant value.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dtype** – Data type of destination, source tensors and **C**.

Remarks

- The destination and source tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.

4.10.9 tpu_bdc_max

Perform maximum operation of the src0 and src1 tensors.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src0_addr** – Address of the src0 tensor.
- **src1_addr** – Address of the src1 tensor.
- **shape** – Pointer to the shape of the destination, src0 and src1 tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src0_stride** – Pointer to the stride of the src0 tensor.
- **src1_stride** – Pointer to the stride of the src1 tensor.
- **dtype** – Data type of the destination, src0 and src1 tensors.

Remarks

- The destination, src0 and src1 tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.

4.10.10 tpu_bdc_max_C

Perform maximum operation of the source tensor and a constant value.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **C** – Constant value.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dtype** – Data type of destination, source tensors and **C**.

Remarks

- The destination and source tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.

4.10.11 tpu_bdc_arithmetic_shift

Perform arithmetic shift operation of the source tensor by the shift tensor.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{src}(n, c, h, w) \text{ LSH } \text{shift}(n, c, h, w) & \text{if } \text{shift}(n, c, h, w) > 0 \\ \text{src}(n, c, h, w) \text{ RSH } - \text{shift}(n, c, h, w) & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **shift_addr** – Address of the shift tensor. Address of the shift tensor.
- **shape** – Pointer to the shape of the destination, source and shift tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **shift_stride** – Pointer to the stride of the shift tensor.
- **dst_dtype** – Data type of destination tensor.
- **src_dtype** – Data type of source tensor.
- **shift_dtype** – Data type of shift tensor.
- **rounding_mode** – Rounding mode for right shift.

Remarks

- The destination, source and shift tensors start at the same NPU.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- **dst_dtype** and **src_dtype** are both in *DT_INT32*, *DT_UINT32*, *DT_INT16*, *DT_UINT16*, *DT_INT8* and *DT_UINT8*. Both are signed or both are unsigned.
- **shift_dtype** is in *DT_INT32*, *DT_INT16* and *DT_INT8*. The element of shift tensors is in [-32, 31].
- The bit width of **src_dtype** is greater than or equal to that of **shift_dtype**.

4.10.12 tpu_bdc_arithmetic_shift_C

Perform arithmetic shift operation of the source tensor by a constant value.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{src}(n, c, h, w) \text{ LSH } C & \text{if } C > 0 \\ \text{src}(n, c, h, w) \text{ RSH } - C & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **C** – Constant value.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dst_dtype** – Data type of destination tensor.
- **src_dtype** – Data type of source tensor.
- **rounding_mode** – Rounding mode for right shift.

Remarks

- The destination and source tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- `dst_dtype` and `src_dtype` are both in `DT_INT32`, `DT_UINT32`, `DT_INT16`, `DT_UINT16`, `DT_INT8` and `DT_UINT8`. Both are signed or both are unsigned.
- `C` is in [-32, 31].

4.10.13 tpu_bdc_logical_shift

Perform logical shift operation of the source tensor by the shift tensor.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{src}(n, c, h, w) \text{ LSH } \text{shift}(n, c, h, w) & \text{if } \text{shift}(n, c, h, w) > 0 \\ \text{src}(n, c, h, w) \text{ RSH } - \text{shift}(n, c, h, w) & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **shift_addr** – Address of the shift tensor.
- **shape** – Pointer to the shape of the destination, source and shift tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **shift_stride** – Pointer to the stride of the shift tensor.
- **dst_dtype** – Data type of destination tensor.
- **src_dtype** – Data type of source tensor.
- **shift_dtype** – Data type of shift tensor.
- **rounding_mode** – Rounding mode for right shift.

Remarks

- The destination, source and shift tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- `dst_dtype` and `src_dtype` are both in *DT_UINT32*, *DT_UINT16* and *DT_UINT8*.
- `shift_dtype` is in *DT_INT32*, *DT_INT16* and *DT_INT8*. The element of shift tensors is in [-32, 31].
- The bit width of `src_dtype` is greater than or equal to that of `shift_dtype`.

4.10.14 tpu_bdc_logical_shift_C

Perform logical shift operation of the source tensor by a constant value.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{src}(n, c, h, w) \text{ LSH } C & \text{if } C > 0 \\ \text{src}(n, c, h, w) \text{ RSH } -C & \text{Otherwise.} \end{cases}$$

Parameters

- `dst_addr` – Address of the destination tensor.
- `src_addr` – Address of the source tensor.
- `C` – Constant value.
- `shape` – Pointer to the shape of the destination and source tensors.
- `dst_stride` – Pointer to the stride of the destination tensor.
- `src_stride` – Pointer to the stride of the source tensor.
- `dst_dtype` – Data type of destination tensor.
- `src_dtype` – Data type of source tensor.
- `rounding_mode` – Rounding mode for right shift.

Remarks

- The destination and source tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- `dst_dtype` and `src_dtype` are both in *DT_UINT32*, *DT_UINT16* and *DT_UINT8*.
- `C` is in [-32, 31].

4.10.15 tpu_bdc_greater

Compare whether the src0 is greater than src1, and then select a true value or zero to be the result.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{true_val} & \text{if } \text{src0}(n, c, h, w) > \text{src1}(n, c, h, w) \\ 0 & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src0_addr** – Address of the src0 tensor.
- **src1_addr** – Address of the src1 tensor.
- **true_val** – True value.
- **shape** – Pointer to the shape of the destination, src0 and src1 tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src0_stride** – Pointer to the stride of the src0 tensor.
- **src1_stride** – Pointer to the stride of the src1 tensor.
- **dst_dtype** – Data type of destination tensor and *true_val*.
- **src_dtype** – Data type of source tensor.

Remarks

- The destination, src0 and src1 tensors start at the same NPU.
- *shape->n*, *shape->c*, *shape->h* and *shape->w* are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The bit width of **src_dtype** is greater than or equal to that of **dst_dtype**.

4.10.16 tpu_bdc_greater_C

Compare whether the source tensor is greater than a constant value, and then select a true value or zero to be the result.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{true_val} & \text{if } \text{src}(n, c, h, w) > C \\ 0 & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **C** – Constant value.
- **true_val** – True value.

- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dst_dtype** – Data type of destination tensor and *true_val*.
- **src_dtype** – Data type of source tensor and *C*.

Remarks

- The destination and source tensors start at the same NPU.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The bit width of **src_dtype** is greater than or equal to that of **dst_dtype**.

4.10.17 tpu_bdc_less

Compare whether the src0 is less than the src1, and then select a true value or zero to be the result.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{true_val} & \text{if } \text{src0}(n, c, h, w) < \text{src1}(n, c, h, w) \\ 0 & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src0_addr** – Address of the src0 tensor.
- **src1_addr** – Address of the src1 tensor.
- **true_val** – True value.
- **shape** – Pointer to the shape of the destination, src0 and src1 tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src0_stride** – Pointer to the stride of the src0 tensor.
- **src1_stride** – Pointer to the stride of the src1 tensor.
- **dst_dtype** – Data type of destination tensor and *true_val*.
- **src_dtype** – Data type of source tensor.

Remarks

- The destination, src0 and src1 tensors start at the same NPU.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The bit width of **src_dtype** is greater than or equal to that of **dst_dtype**.

4.10.18 tpu_bdc_less_C

Compare whether the source tensor is less than a constant value, and then select a true value or zero to be the result.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{true_val} & \text{if } \text{src}(n, c, h, w) < C \\ 0 & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **C** – Constant value.
- **true_val** – True value.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dst_dtype** – Data type of destination tensor and *true_val*.
- **src_dtype** – Data type of source tensor and *C*. Data type of source tensor and *C*.

Remarks

- The destination and source tensors start at the same NPU.
- *shape->n*, *shape->c*, *shape->h* and *shape->w* are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The bit width of *src_dtype* is greater than or equal to that of *dst_dtype*.

4.10.19 tpu_bdc_equal

Compare whether the src0 is equal to the src1, and then select a true value or zero to be the result.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{true_val} & \text{if } \text{src0}(n, c, h, w) = \text{src1}(n, c, h, w) \\ 0 & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src0_addr** – Address of the src0 tensor.
- **src1_addr** – Address of the src1 tensor.
- **true_val** – True value.
- **shape** – Pointer to the shape of the destination, src0 and src1 tensors.

- **dst_stride** – Pointer to the stride of the destination tensor.
- **src0_stride** – Pointer to the stride of the src0 tensor.
- **src1_stride** – Pointer to the stride of the src1 tensor.
- **dst_dtype** – Data type of destination tensor and *true_val*.
- **src_dtype** – Data type of source tensor.

Remarks

- The destination, src0 and src1 tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The bit width of **src_dtype** is greater than or equal to that of **dst_dtype**.

4.10.20 tpu_bdc_equal_C

Compare whether the source tensor is equal to a constant value, and then select a true value or zero to be the result.

```
void tpu_bdc_equal_C(local_addr_t dst_addr, local_addr_t src_addr, scalar_t C,
                    scalar_t true_val, const dim4_t *shape, const dim4_t *dst_stride,
                    const dim4_t *src_stride, data_type_t dst_dtype, data_type_t
                    src_dtype)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{true_val} & \text{if } \text{src}(n, c, h, w) = C \\ 0 & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **C** – Constant value.
- **true_val** – True value.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dst_dtype** – Data type of destination tensor and *true_val*.
- **src_dtype** – Data type of source tensor and *C*.

Remarks

- The destination and source tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The bit width of **src_dtype** is greater than or equal to that of **dst_dtype**.

4.10.21 tpu_bdc_greater_equal

Compare whether the src0 is greater than or equal to the src1, and then select a true value or zero to be the result.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{true_val} & \text{if } \text{src0}(n, c, h, w) \geq \text{src1}(n, c, h, w) \\ 0 & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src0_addr** – Address of the src0 tensor.
- **src1_addr** – Address of the src1 tensor.
- **true_val** – True value.
- **shape** – Pointer to the shape of the destination, src0 and src1 tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src0_stride** – Pointer to the stride of the src0 tensor.
- **src1_stride** – Pointer to the stride of the src1 tensor.
- **dst_dtype** – Data type of destination tensor and *true_val*.
- **src_dtype** – Data type of source tensor.

Remarks

- The destination, src0 and src1 tensors start at the same NPU.
- *shape->n*, *shape->c*, *shape->h* and *shape->w* are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The bit width of *src_dtype* is greater than or equal to that of *dst_dtype*.

4.10.22 tpu_bdc_greater_equal_C

Compare whether the source tensor is greater than or equal to a constant value, and then select a true value or zero to be the result.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{true_val} & \text{if } \text{src}(n, c, h, w) \geq C \\ 0 & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **C** – Constant value.

- **true_val** – True value.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dst_dtype** – Data type of destination tensor and *true_val*.
- **src_dtype** – Data type of source tensor and *C*.

Remarks

- The destination and source tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The bit width of `src_dtype` is greater than or equal to that of `dst_dtype`.

4.10.23 tpu_bdc_less_equal

Compare whether the `src0` is less than or equal to the `src1`, and then select a true value or zero to be the result.

```
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 src_dtype)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{true_val} & \text{if } \text{src0}(n, c, h, w) \leq \text{src1}(n, c, h, w) \\ 0 & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src0_addr** – Address of the `src0` tensor.
- **src1_addr** – Address of the `src1` tensor.
- **true_val** – True value.
- **shape** – Pointer to the shape of the destination, `src0` and `src1` tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src0_stride** – Pointer to the stride of the `src0` tensor.
- **src1_stride** – Pointer to the stride of the `src1` tensor.
- **dst_dtype** – Data type of destination tensor and *true_val*.
- **src_dtype** – Data type of source tensor.

Remarks

- The destination, `src0` and `src1` tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The bit width of `src_dtype` is greater than or equal to that of `dst_dtype`.

4.10.24 tpu_bdc_less_equal_C

Compare whether the source tensor is less than or equal to a constant value, and then select a true value or zero to be the result.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{true_val} & \text{if } \text{src}(n, c, h, w) \leq C \\ 0 & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **C** – Constant value.
- **true_val** – True value.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dst_dtype** – Data type of destination tensor and *true_val*.
- **src_dtype** – Data type of source tensor and *C*.

Remarks

- The destination and source tensors start at the same NPU.
- *shape->n*, *shape->c*, *shape->h* and *shape->w* are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The bit width of *src_dtype* is greater than or equal to that of *dst_dtype*.

4.10.25 tpu_bdc_not_equal

Compare whether the src0 is not equal to the src1, and then select a true value or zero to be the result.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{true_val} & \text{if } \text{src0}(n, c, h, w) \neq \text{src1}(n, c, h, w) \\ 0 & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src0_addr** – Address of the src0 tensor.
- **src1_addr** – Address of the src1 tensor.
- **true_val** – True value.

- **shape** – Pointer to the shape of the destination, src0 and src1 tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src0_stride** – Pointer to the stride of the src0 tensor.
- **src1_stride** – Pointer to the stride of the src1 tensor.
- **dst_dtype** – Data type of destination tensor and *true_val*.
- **src_dtype** – Data type of source tensor.

Remarks

- The destination, src0 and src1 tensors start at the same NPU.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The bit width of **src_dtype** is greater than or equal to that of **dst_dtype**.

4.10.26 tpu_bdc_not_equal_C

Compare whether the source tensor is not equal to a constant value, and then select a true value or zero to be the result.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{true_val} & \text{if } \text{src}(n, c, h, w) \neq C \\ 0 & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **C** – Constant value.
- **true_val** – True value.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dst_dtype** – Data type of destination tensor and *true_val*.
- **src_dtype** – Data type of source tensor and *C*.

Remarks

- The destination and source tensors start at the same NPU.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The bit width of **src_dtype** is greater than or equal to that of **dst_dtype**.

4.10.27 tpu_bdc_vc_and

Perform bitwise AND operation of the src0 and src1 vectors.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination matrix.
- **src0_addr** – Address of the src0 vector.
- **src1_addr** – Address of the src1 vector.
- **src0_len** – Length of src0 vector.
- **src1_len** – Length of src1 vector.
- **src0_len_per_channel** – Length of src0 vector in each channel.
- **src1_len_per_channel** – Length of src1 vector in each channel.
- **dtype** – Data type of the destination, src0 and src1.

Remarks

- The destination matrix and src1 vector start at the same NPU.
- **src0_len** and **src1_len** are in [1, 65535].
- The destination matrix is in matrix layout, with rows of **src0_len** and columns of **src1_len**. The src0 and src1 vectors is in vector layout.

4.10.28 tpu_bdc_vc_or

Perform bitwise OR operation of the src0 and src1 vectors.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination matrix.
- **src0_addr** – Address of the src0 vector.
- **src1_addr** – Address of the src1 vector.
- **src0_len** – Length of src0 vector.
- **src1_len** – Length of src1 vector.
- **src0_len_per_channel** – Length of src0 vector in each channel.
- **src1_len_per_channel** – Length of src1 vector in each channel.
- **dtype** – Data type of the destination, src0 and src1.

Remarks

- The destination matrix and src1 vector start at the same NPU.

- `src0_len` and `src1_len` are in [1, 65535].
- The destination matrix is in matrix layout, with rows of `src0_len` and columns of `src1_len`. The `src0` and `src1` vectors is in vector layout.

4.10.29 tpu_bdc_vc_xor

Perform bitwise XOR operation of the `src0` and `src1` vectors.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination matrix.
- **src0_addr** – Address of the `src0` vector.
- **src1_addr** – Address of the `src1` vector.
- **src0_len** – Length of `src0` vector.
- **src1_len** – Length of `src1` vector.
- **src0_len_per_channel** – Length of `src0` vector in each channel.
- **src1_len_per_channel** – Length of `src1` vector in each channel.
- **dtype** – Data type of the destination, `src0` and `src1`.

Remarks

- The destination matrix and `src1` vector start at the same NPU.
- `src0_len` and `src1_len` are in [1, 65535].
- The destination matrix is in matrix layout, with rows of `src0_len` and columns of `src1_len`. The `src0` and `src1` vectors is in vector layout.

4.10.30 tpu_bdc_vc_min

Perform minimization operation of the `src0` and `src1` vectors.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination matrix.
- **src0_addr** – Address of the `src0` vector.
- **src1_addr** – Address of the `src1` vector.
- **src0_len** – Length of `src0` vector.
- **src1_len** – Length of `src1` vector.
- **src0_len_per_channel** – Length of `src0` vector in each channel.

- **src1_len_per_channel** – Length of src1 vector in each channel.
- **dtype** – Data type of the destination, src0 and src1.

Remarks

- The destination matrix and src1 vector start at the same NPU.
- **src0_len** and **src1_len** are in [1, 65535].
- The destination matrix is in matrix layout, with rows of **src0_len** and columns of **src1_len**. The src0 and src1 vectors is in vector layout.

4.10.31 tpu_bdc_vc_max

Perform maximization operation of the src0 and src1 vectors.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination matrix.
- **src0_addr** – Address of the src0 vector.
- **src1_addr** – Address of the src1 vector.
- **src0_len** – Length of src0 vector.
- **src1_len** – Length of src1 vector.
- **src0_len_per_channel** – Length of src0 vector in each channel.
- **src1_len_per_channel** – Length of src1 vector in each channel.
- **dtype** – Data type of the destination, src0 and src1.

Remarks

- The destination matrix and src1 vector start at the same NPU.
- **src0_len** and **src1_len** are in [1, 65535].
- The destination matrix is in matrix layout, with rows of **src0_len** and columns of **src1_len**. The src0 and src1 vectors is in vector layout.

4.10.32 tpu_bdc_vc_greater

Compare whether the src0 is greater than the src1, and then select a true value or zero to be the result.

```
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)
```

$$\text{dst}(m, n) = \begin{cases} \text{true_val} & \text{if } \text{src0}(m) > \text{src1}(n) \\ 0 & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination matrix.
- **src0_addr** – Address of the src0 vector.
- **src1_addr** – Address of the src1 vector.
- **true_val** – True value.
- **src0_len** – Length of src0 vector.
- **src1_len** – Length of src1 vector.
- **src0_len_per_channel** – Length of src0 vector in each channel.
- **src1_len_per_channel** – Length of src1 vector in each channel.
- **dst_dtype** – Data type of destination matrix and *true_val*.
- **src_dtype** – Data type of source vector.

Remarks

- The destination matrix and src1 vector start at the same NPU.
- **src0_len** and **src1_len** are in [1, 65535].
- The destination matrix is in matrix layout, with rows of **src0_len** and columns of **src1_len**. The src0 and src1 vectors is in vector layout.
- The bit width of **src_dtype** is greater than or equal to that of **dst_dtype**.

4.10.33 tpu_bdc_vc_less

Compare whether the src0 is less than the src1, and then select a true value or zero to be the result.

```
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)
```

$$\text{dst}(m, n) = \begin{cases} \text{true_val} & \text{if } \text{src0}(m) < \text{src1}(n) \\ 0 & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination matrix.
- **src0_addr** – Address of the src0 vector.
- **src1_addr** – Address of the src1 vector.
- **true_val** – True value.
- **src0_len** – Length of src0 vector.
- **src1_len** – Length of src1 vector.
- **src0_len_per_channel** – Length of src0 vector in each channel.
- **src1_len_per_channel** – Length of src1 vector in each channel.
- **dst_dtype** – Data type of destination matrix and *true_val*.
- **src_dtype** – Data type of source vector.

Remarks

- The destination matrix and src1 vector start at the same NPU.
- **src0_len** and **src1_len** are in [1, 65535].

- The destination matrix is in matrix layout, with rows of `src0_len` and columns of `src1_len`. The `src0` and `src1` vectors is in vector layout.
- The bit width of `src_dtype` is greater than or equal to that of `dst_dtype`.

4.10.34 tpu_bdc_vc_equal

Compare whether the `src0` is equal to the `src1`, and then select a true value or zero to be the result.

```
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)
```

$$\text{dst}(m, n) = \begin{cases} \text{true_val} & \text{if } \text{src0}(m) = \text{src1}(n) \\ 0 & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination matrix.
- **src0_addr** – Address of the `src0` vector.
- **src1_addr** – Address of the `src1` vector.
- **true_val** – True value.
- **src0_len** – Length of `src0` vector.
- **src1_len** – Length of `src1` vector.
- **src0_len_per_channel** – Length of `src0` vector in each channel.
- **src1_len_per_channel** – Length of `src1` vector in each channel.
- **dst_dtype** – Data type of destination matrix and *true_val*.
- **src_dtype** – Data type of source vector.

Remarks

- The destination matrix and `src1` vector start at the same NPU.
- `src0_len` and `src1_len` are in [1, 65535].
- The destination matrix is in matrix layout, with rows of `src0_len` and columns of `src1_len`. The `src0` and `src1` vectors is in vector layout.
- The bit width of `src_dtype` is greater than or equal to that of `dst_dtype`.

4.10.35 tpu_bdc_vc_greater_equal

Compare whether the `src0` is greater than or equal to the `src1`, and then select a true value or zero to be the result.

```
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)
```

$$\text{dst}(m, n) = \begin{cases} \text{true_val} & \text{if } \text{src0}(m) \geq \text{src1}(n) \\ 0 & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination matrix.
- **src0_addr** – Address of the src0 vector.
- **src1_addr** – Address of the src1 vector.
- **true_val** – True value.
- **src0_len** – Length of src0 vector.
- **src1_len** – Length of src1 vector.
- **src0_len_per_channel** – Length of src0 vector in each channel.
- **src1_len_per_channel** – Length of src1 vector in each channel.
- **dst_dtype** – Data type of destination matrix and *true_val*.
- **src_dtype** – Data type of source vector.

Remarks

- The destination matrix and src1 vector start at the same NPU.
- **src0_len** and **src1_len** are in [1, 65535].
- The destination matrix is in matrix layout, with rows of **src0_len** and columns of **src1_len**. The src0 and src1 vectors is in vector layout.
- The bit width of **src_dtype** is greater than or equal to that of **dst_dtype**.

4.10.36 tpu_bdc_vc_less_equal

Compare whether the src0 is less than or equal to the src1, and then select a true value or zero to be the result.

```
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)
```

$$\text{dst}(m, n) = \begin{cases} \text{true_val} & \text{if } \text{src0}(m) \leq \text{src1}(n) \\ 0 & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination matrix.
- **src0_addr** – Address of the src0 vector.
- **src1_addr** – Address of the src1 vector.
- **true_val** – True value.
- **src0_len** – Length of src0 vector.
- **src1_len** – Length of src1 vector.
- **src0_len_per_channel** – Length of src0 vector in each channel.
- **src1_len_per_channel** – Length of src1 vector in each channel.
- **dst_dtype** – Data type of destination matrix and *true_val*.
- **src_dtype** – Data type of source vector.

Remarks

- The destination matrix and src1 vector start at the same NPU.
- `src0_len` and `src1_len` are in [1, 65535].
- The destination matrix is in matrix layout, with rows of `src0_len` and columns of `src1_len`. The src0 and src1 vectors is in vector layout.
- The bit width of `src_dtype` is greater than or equal to that of `dst_dtype`.

4.10.37 tpu_bdc_vc_not_equal

Compare whether the src0 is not equal to the src1, and then select a true value or zero to be the result.

```
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)
```

$$\text{dst}(m, n) = \begin{cases} \text{true_val} & \text{if } \text{src0}(m) \neq \text{src1}(n) \\ 0 & \text{Otherwise.} \end{cases}$$

Parameters

- `dst_addr` – Address of the destination matrix.
- `src0_addr` – Address of the src0 vector.
- `src1_addr` – Address of the src1 vector.
- `true_val` – True value.
- `src0_len` – Length of src0 vector.
- `src1_len` – Length of src1 vector.
- `src0_len_per_channel` – Length of src0 vector in each channel.
- `src1_len_per_channel` – Length of src1 vector in each channel.
- `dst_dtype` – Data type of destination matrix and *true_val*.
- `src_dtype` – Data type of source vector.

Remarks

- The destination matrix and src1 vector start at the same NPU.
- `src0_len` and `src1_len` are in [1, 65535].
- The destination matrix is in matrix layout, with rows of `src0_len` and columns of `src1_len`. The src0 and src1 vectors is in vector layout.
- The bit width of `src_dtype` is greater than or equal to that of `dst_dtype`.

4.11 Floating Point Binary Functions

4.11.1 tpu_bdc_fp_add

Perform addition of the src0 and src1 tensors.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src0_addr** – Address of the src0 tensor.
- **src1_addr** – Address of the src1 tensor.
- **shape** – Pointer to the shape of the destination, src0 and src1 tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src0_stride** – Pointer to the stride of the src0 tensor.
- **src1_stride** – Pointer to the stride of the src1 tensor.
- **dtype** – Data type of the destination, src0 and src1 tensors.

Remarks

- The destination, src0 and src1 tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The valid choices of **dtype** are *DT_FP32*, *DT_FP16* and *DT_BFP16*.

4.11.2 tpu_bdc_fp_add_C

Perform addition of the source tensor and a constant value.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **C** – Constant value.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dtype** – Data type of destination, source tensors and *C*.

Remarks

- The destination and source tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The valid choices of `dtype` are `DT_FP32`, `DT_FP16` and `DT_BFP16`.

4.11.3 tpu_bdc_fp_sub

Perform subtraction of the src0 and src1 tensors.

```
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)
```

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

Parameters

- `dst_addr` – Address of the destination tensor.
- `src0_addr` – Address of the src0 tensor.
- `src1_addr` – Address of the src1 tensor.
- `shape` – Pointer to the shape of the destination, src0 and src1 tensors.
- `dst_stride` – Pointer to the stride of the destination tensor.
- `src0_stride` – Pointer to the stride of the src0 tensor.
- `src1_stride` – Pointer to the stride of the src1 tensor.
- `dtype` – Data type of the destination, src0 and src1 tensors.

Remarks

- The destination, src0 and src1 tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The valid choices of `dtype` are `DT_FP32`, `DT_FP16` and `DT_BFP16`.

4.11.4 tpu_bdc_fp_sub_C

Perform subtraction of the source tensor by a constant value.

```
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)
```

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

Parameters

- `dst_addr` – Address of the destination tensor.
- `src_addr` – Address of the source tensor.
- `C` – Constant value.

- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dtype** – Data type of destination, source tensors and \mathcal{C} .

Remarks

- The destination and source tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The valid choices of **dtype** are *DT_FP32*, *DT_FP16* and *DT_BFP16*.

4.11.5 tpu_bdc_fp_C_sub

Perform subtraction of a constant value by the source tensor.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **C** – Constant value.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dtype** – Data type of destination, source tensors and \mathcal{C} .

Remarks

- The destination and source tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The valid choices of **dtype** are *DT_FP32*, *DT_FP16* and *DT_BFP16*.

4.11.6 tpu_bdc_fp_mul

Perform multiplication of the src0 and src1 tensors.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src0_addr** – Address of the src0 tensor.
- **src1_addr** – Address of the src1 tensor.
- **shape** – Pointer to the shape of the destination, src0 and src1 tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src0_stride** – Pointer to the stride of the src0 tensor.
- **src1_stride** – Pointer to the stride of the src1 tensor.
- **dtype** – Data type of the destination, src0 and src1 tensors.

Remarks

- The destination, src0 and src1 tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The valid choices of `dtype` are `DT_FP32`, `DT_FP16` and `DT_BFP16`.

4.11.7 tpu_bdc_fp_mul_C

Perform multiplication of the source tensor and a constant value.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **C** – Constant value.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dtype** – Data type of destination, source tensors and `C`.

Remarks

- The destination and source tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The valid choices of `dtype` are `DT_FP32`, `DT_FP16` and `DT_BFP16`.

4.11.8 tpu_bdc_fp32_div

Perform division of the src0 tensor by the src1 tensor.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src0_addr** – Address of the src0 tensor.
- **src1_addr** – Address of the src1 tensor.
- **shape** – Pointer to the shape of the destination, src0 and src1 tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src0_stride** – Pointer to the stride of the src0 tensor.
- **src1_stride** – Pointer to the stride of the src1 tensor.

Remarks

- The destination, src0 and src1 tensors start at the same NPU.
- shape->n, shape->c, shape->h and shape->w are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- It is equivalent to `tpu_bdc_fp32_tunable_div()` with a Newton iteration of 3.

4.11.9 tpu_bdc_fp32_div_C

Perform division of the source tensor by a constant value.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **C** – Constant value.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.

Remarks

- The destination and source tensors start at the same NPU.
- shape->n, shape->c, shape->h and shape->w are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- It is equivalent to `tpu_bdc_fp32_tunable_div_C()` with a Newton iteration of 3.

4.11.10 tpu_bdc_fp32_C_div

Perform division of a constant value by the source tensor.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **C** – Constant value.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.

Remarks

- The destination and source tensors start at the same NPU.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- It is equivalent to `tpu_bdc_fp32_tunable_C_div()` with a Newton iteration of 3.

4.11.11 tpu_bdc_fp32_tunable_div

Perform division of the src0 tensor by the src1 tensor.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src0_addr** – Address of the src0 tensor.
- **src1_addr** – Address of the src1 tensor.
- **shape** – Pointer to the shape of the destination, src0 and src1 tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src0_stride** – Pointer to the stride of the src0 tensor.
- **src1_stride** – Pointer to the stride of the src1 tensor.
- **num_iter** – Newton iterations.

Remarks

- The destination, src0 and src1 tensors start at the same NPU.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535].

- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- `num_iter` is a performance parameter, in [1, 4].

4.11.12 `tpu_bdc_fp32_tunable_div_C`

Perform division of the source tensor by a constant value.

```
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)
```

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

Parameters

- `dst_addr` – Address of the destination tensor.
- `src_addr` – Address of the source tensor.
- `C` – Constant value.
- `shape` – Pointer to the shape of the destination and source tensors.
- `dst_stride` – Pointer to the stride of the destination tensor.
- `src_stride` – Pointer to the stride of the source tensor.
- `num_iter` – Newton iterations.

Remarks

- The destination and source tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- `num_iter` is a performance parameter, in [1, 4].

4.11.13 `tpu_bdc_fp32_tunable_C_div`

Perform division of a constant value by the source tensor.

```
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)
```

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

Parameters

- `dst_addr` – Address of the destination tensor.
- `src_addr` – Address of the source tensor.
- `C` – Constant value.
- `shape` – Pointer to the shape of the destination and source tensors.
- `dst_stride` – Pointer to the stride of the destination tensor.
- `src_stride` – Pointer to the stride of the source tensor.
- `num_iter` – Newton iterations.

Remarks

- The destination and source tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- `num_iter` is a performance parameter, in [1, 4].

4.11.14 tpu_bdc_fp32_mac

Perform multiplication between the `src0` and `src1` tensors, and accumulate results.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src0_addr** – Address of the `src0` tensor.
- **src1_addr** – Address of the `src1` tensor.
- **shape** – Pointer to the shape of the destination, `src0` and `src1` tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src0_stride** – Pointer to the stride of the `src0` tensor.
- **src1_stride** – Pointer to the stride of the `src1` tensor.

Remarks

- The destination, `src0` and `src1` tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.

4.11.15 tpu_bdc_fp32_mac_C

Perform multiplication between the source tensor and a constant, and accumulate results.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **C** – Constant value.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.

Remarks

- The destination and source tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.

4.11.16 tpu_bdc_fp_diff_abs

Compute the absolute value of the element-wise difference between the src0 and src1 tensors.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src0_addr** – Address of the src0 tensor.
- **src1_addr** – Address of the src1 tensor.
- **shape** – Pointer to the shape of the destination, src0 and src1 tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src0_stride** – Pointer to the stride of the src0 tensor.
- **src1_stride** – Pointer to the stride of the src1 tensor.
- **dtype** – Data type of the destination, src0 and src1 tensors.

Remarks

- The destination, src0 and src1 tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The valid choices of dtype are *DT_FP32*, *DT_FP16* and *DT_BFP16*.

4.11.17 tpu_bdc_fp_diff_abs_C

Compute the absolute value of the difference between the source tensor and a constant value.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **C** – Constant value.
- **shape** – Pointer to the shape of the destination and source tensors.

- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dtype** – Data type of destination, source tensors and *C*.

Remarks

- The destination and source tensors start at the same NPU.
- *shape->n*, *shape->c*, *shape->h* and *shape->w* are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The valid choices of **dtype** are *DT_FP32*, *DT_FP16* and *DT_BFP16*.

4.11.18 tpu_bdc_fp32_pow

Compute the power of src0 tensor and use src1 tensor as exponent.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src0_addr** – Address of the src0 tensor.
- **src1_addr** – Address of the src1 tensor.
- **work0_addr** – Address of the work0 tensor.
- **work1_addr** – Address of the work1 tensor.
- **exp_coeff_addr** – Address of the exp_coeff.
- **log_coeff_addr** – Address of the log_coeff.
- **exp_table_addr** – Address of the exp_table.
- **shape** – Pointer to the shape of the destination, src0, src1, work0 and work1 tensors.

Remarks

- The destination, src0, src1, work0 and work1 tensors start at the same NPU, and all are in 64-byte aligned layout.
- *exp_coeff* is loaded by *tpu_bdc_load_fp32_exp_coeff()*. *log_coeff* is loaded by *tpu_bdc_load_fp32_log_coeff()*. *exp_table* is loaded by *tpu_bdc_load_fp32_exp_table()*.
- The work0 and work1 tensors are workspaces to store temporary tensors. Any two of *dst_addr*, *work0_addr* and *work1_addr* are not allowed to be equal. The *src0_addr* is equal to *dst_addr*, *work0_addr* or *work1_addr*, in this case, data of src0 will be overwritten. The *src1_addr* is equal to *dst_addr*, *work0_addr* or *work1_addr*, in this case, data of src1 will be overwritten. The *src0_addr* is equal to *src0_addr*. If there is no bank conflicting between any two of destination, work1 and table tensors, the performance will be better.
- *shape->n*, *shape->c*, *shape->h* and *shape->w* are in [1, 65535], *shape->h* * *shape->w* less than or equal to 65535.

- The elements of `src0` is in $(0, +\infty)$. If $\text{src1}(n, c, h, w) \times \log(\text{src0}(n, c, h, w))$ is less than -103 or greater than 88, then $\text{dst}(n, c, h, w)$ will be e^{-103} or e^{88} , respectively.

4.11.19 tpu_bdc_fp32_pow_C

Compute the power of `src0` tensor and use a constant value as exponent.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **work0_addr** – Address of the work0 tensor.
- **work1_addr** – Address of the work1 tensor.
- **exp_coeff_addr** – Address of the `exp_coeff`.
- **log_coeff_addr** – Address of the `log_coeff`.
- **exp_table_addr** – Address of the `exp_table`.
- **C** – Constant value.
- **shape** – Pointer to the shape of the destination, source, work0 and work1 tensors.

Remarks

- The destination, source, work0 and work1 tensors start at the same NPU, and all are in 64-byte aligned layout.
- `exp_coeff` is loaded by `tpu_bdc_load_fp32_exp_coeff()`. `log_coeff` is loaded by `tpu_bdc_load_fp32_log_coeff()`. `exp_table` is loaded by `tpu_bdc_load_fp32_exp_table()`.
- The work0 and work1 tensors are workspaces to store temporary tensors. Any two of `dst_addr`, `work0_addr` and `work1_addr` are not allowed to be equal. The `src_addr` is equal to `dst_addr`, `work0_addr` or `work1_addr`, in this case, source data will be overwritten. If there is no bank conflicting between any two of destination, work1 and table tensors, the performance will be better.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in $[1, 65535]$, `shape->h * shape->w` less than or equal to 65535.
- The elements of `src` is in $(0, +\infty)$. If $C \times \log(\text{src}(n, c, h, w))$ is less than -103 or greater than 88, then $\text{dst}(n, c, h, w)$ will be e^{-103} or e^{88} , respectively.

4.11.20 tpu_bdc_fp32_C_pow

Compute the power of the constant value and use the source tensor as exponent.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **work0_addr** – Address of the work0 tensor.
- **work1_addr** – Address of the work1 tensor.
- **exp_coeff_addr** – Address of the exp_coeff.
- **exp_table_addr** – Address of the exp_table.
- **C** – Constant value.
- **shape** – Pointer to the shape of the destination, source, work0 and work1 tensors.

Remarks

- The destination, source, work0 and work1 tensors start at the same NPU, and all are in 64-byte aligned layout.
- exp_coeff is loaded by `tpu_bdc_load_fp32_exp_coeff()`. exp_table is loaded by `tpu_bdc_load_fp32_exp_table()`.
- The work0 and work1 tensors are workspaces to store temporary tensors. Any two of dst_addr, work0_addr and work1_addr are not allowed to be equal. The src_addr is equal to dst_addr, work0_addr or work1_addr, in this case, source data will be overwritten. If there is no bank conflicting between any two of destination, work1 and table tensors, the performance will be better.
- shape->n, shape->c, shape->h and shape->w are in [1, 65535], shape->h * shape->w less than or equal to 65535.
- C is in (0, +∞). If $\text{src}(n, c, h, w) \times \log(C)$ is less than -103 or greater than 88, then $\text{dst}(n, c, h, w)$ will be e^{-103} or e^{88} , respectively.

4.11.21 tpu_bdc_fp_vc_add

Perform addition of the src0 and src1 vectors.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination matrix.
- **src0_addr** – Address of the src0 vector.

- **src1_addr** – Address of the src1 vector.
- **src0_len** – Length of src0 vector.
- **src1_len** – Length of src1 vector.
- **src0_len_per_channel** – Length of src0 vector in each channel.
- **src1_len_per_channel** – Length of src1 vector in each channel.
- **dtype** – Data type of the destination, src0 and src1.

Remarks

- The destination matrix and src1 vector start at the same NPU.
- **src0_len** and **src1_len** are in [1, 65535].
- The destination matrix is in matrix layout, with rows of **src0_len** and columns of **src1_len**. The src0 and src1 vectors is in vector layout.
- The valid choices of **dtype** are *DT_FP32*, *DT_FP16* and *DT_BFP16*.

4.11.22 tpu_bdc_fp_vc_sub

Perform subtraction of the src0 and src1 vectors.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination matrix.
- **src0_addr** – Address of the src0 vector.
- **src1_addr** – Address of the src1 vector.
- **src0_len** – Length of src0 vector.
- **src1_len** – Length of src1 vector.
- **src0_len_per_channel** – Length of src0 vector in each channel.
- **src1_len_per_channel** – Length of src1 vector in each channel.
- **dtype** – Data type of the destination, src0 and src1.

Remarks

- The destination matrix and src1 vector start at the same NPU.
- **src0_len** and **src1_len** are in [1, 65535].
- The destination matrix is in matrix layout, with rows of **src0_len** and columns of **src1_len**. The src0 and src1 vectors is in vector layout.
- The valid choices of **dtype** are *DT_FP32*, *DT_FP16* and *DT_BFP16*.

4.11.23 tpu_bdc_fp_vc_mul

Perform multiplication of the src0 and src1 vectors.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination matrix.
- **src0_addr** – Address of the src0 vector.
- **src1_addr** – Address of the src1 vector.
- **src0_len** – Length of src0 vector.
- **src1_len** – Length of src1 vector.
- **src0_len_per_channel** – Length of src0 vector in each channel.
- **src1_len_per_channel** – Length of src1 vector in each channel.
- **dtype** – Data type of the destination, src0 and src1.

Remarks

- The destination matrix and src1 vector start at the same NPU.
- **src0_len** and **src1_len** are in [1, 65535].
- The destination matrix is in matrix layout, with rows of **src0_len** and columns of **src1_len**. The src0 and src1 vectors is in vector layout.
- The valid choices of **dtype** are *DT_FP32*, *DT_FP16* and *DT_BFP16*.

4.11.24 tpu_bdc_fp32_vc_div

Perform division of the src0 vector by the src1 vector.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination matrix.
- **src0_addr** – Address of the src0 vector.
- **src1_addr** – Address of the src1 vector.
- **src0_len** – Length of src0 vector.
- **src1_len** – Length of src1 vector.
- **src0_len_per_channel** – Length of src0 vector in each channel.
- **src1_len_per_channel** – Length of src1 vector in each channel.

Remarks

- The destination matrix and src1 vector start at the same NPU.
- `src0_len` and `src1_len` are in [1, 65535].
- The destination matrix is in matrix layout, with rows of `src0_len` and columns of `src1_len`. The src0 and src1 vectors is in vector layout.

4.12 Fixed Point Binary Functions

4.12.1 tpu_bdc_int_add

Add the src0 and src1 tensors, then perform arithmetic shift and saturation (optional) on the result.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (\text{src0}(n, c, h, w) + \text{src1}(n, c, h, w)) \text{ LSH } \text{shift} & \text{if shift} > 0 \\ (\text{src0}(n, c, h, w) + \text{src1}(n, c, h, w)) \text{ RSH } - \text{shift} & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src0_addr** – Address of the src0 tensor.
- **src1_addr** – Address of the src1 tensor.
- **shape** – Pointer to the shape of the destination, src0 and src1 tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src0_stride** – Pointer to the stride of the src0 tensor.
- **src1_stride** – Pointer to the stride of the src1 tensor.
- **dst_dtype** – Data type of the destination tensor.
- **src0_dtype** – Data type of the src0 tensor.
- **src1_dtype** – Data type of the src1 tensor.
- **shift** – Number of shifts.
- **rounding_mode** – Rounding mode for right shift.
- **saturation** – Flag of saturation.

Remarks

- The destination, src0 and src1 tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The `shift` is in [-32, 31].
- The valid choices of `dst_dtype`, `src0_dtype` and `src1_dtype` are `DT_INT32`, `DT_UINT32`, `DT_INT16`, `DT_UINT16`, `DT_INT8` and `DT_UINT8`. `dst_dtype` is unsigned only when `src0_dtype` and `src1_dtype` are both unsigned.

4.12.2 tpu_bdc_int_add_C

Add the source tensor and constant value, then perform arithmetic shift and saturation (optional) on the result.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (\text{src}(n, c, h, w) + C) \text{ LSH } \text{shift} & \text{if shift} > 0 \\ (\text{src}(n, c, h, w) + C) \text{ RSH } - \text{shift} & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **C** – Constant value.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dst_dtype** – Data type of the destination tensor.
- **src_dtype** – Data type of source tensor.
- **C_dtype** – Data type of *C*.
- **shift** – Number of shifts.
- **rounding_mode** – Rounding mode for right shift.
- **saturation** – Flag of saturation.

Remarks

- The destination and source tensors start at the same NPU.
- *shape->n*, *shape->c*, *shape->h* and *shape->w* are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The *shift* is in [-32, 31].
- The valid choices of *dst_dtype*, *src_dtype* and *C_dtype* are *DT_INT32*, *DT_UINT32*, *DT_INT16*, *DT_UINT16*, *DT_INT8* and *DT_UINT8*. *dst_dtype* is unsigned only when *src_dtype* and *C_dtype* are both unsigned.

4.12.3 tpu_bdc_int_pcs_add

Add the src0 and src1 tensors, then perform arithmetic shift by channel and saturation (optional) on the result.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (\text{src0}(n, c, h, w) + \text{src1}(n, c, h, w)) \text{ LSH } \text{shift}(0, c, 0, 0) & \text{if } \text{shift}(0, c, 0, 0) > 0 \\ (\text{src0}(n, c, h, w) + \text{src1}(n, c, h, w)) \text{ RSH } - \text{shift}(0, c, 0, 0) & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src0_addr** – Address of the src0 tensor.
- **src1_addr** – Address of the src1 tensor.
- **shift_addr** – Address of the shift tensor.
- **shape** – Pointer to the shape of the destination, src0 and src1 tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src0_stride** – Pointer to the stride of the src0 tensor.
- **src1_stride** – Pointer to the stride of the src1 tensor.
- **dst_dtype** – Data type of the destination tensor.
- **src0_dtype** – Data type of the src0 tensor.
- **src1_dtype** – Data type of the src1 tensor.
- **rounding_mode** – Rounding mode for right shift.
- **saturation** – Flag of saturation.

Remarks

- The destination, src0, src1 and shift start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The shape of shift is [1, `shape->c`, 1, 1], in compact layout. The data type of shift is `DT_INT8`, and the value range is [-32, 31].
- The valid choices of `dst_dtype`, `src0_dtype` and `src1_dtype` are `DT_INT32`, `DT_UINT32`, `DT_INT16`, `DT_UINT16`, `DT_INT8` and `DT_UINT8`. `dst_dtype` is unsigned only when `src0_dtype` and `src1_dtype` are both unsigned.

4.12.4 tpu_bdc_int_pcs_add_C

Add the source tensor and constant value, then perform arithmetic shift by channel and saturation (optional) on the result.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (\text{src}(n, c, h, w) + C) \text{ LSH } \text{shift}(0, c, 0, 0) & \text{if } \text{shift}(0, c, 0, 0) > 0 \\ (\text{src}(n, c, h, w) + C) \text{ RSH } - \text{shift}(0, c, 0, 0) & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **shift_addr** – Address of the shift tensor.
- **C** – Constant value.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dst_dtype** – Data type of the destination tensor.
- **src_dtype** – Data type of source tensor.
- **C_dtype** – Data type of *C*.
- **rounding_mode** – Rounding mode for right shift.
- **saturation** – Flag of saturation.

Remarks

- The destination, src and shift start at the same NPU.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The shape of shift is [1, **shape->c**, 1, 1], in compact layout. The data type of shift is *DT_INT8*, and the value range is [-32, 31].
- The valid choices of **dst_dtype**, **src_dtype** and **C_dtype** are *DT_INT32*, *DT_UINT32*, *DT_INT16*, *DT_UINT16*, *DT_INT8* and *DT_UINT8*. **dst_dtype** is unsigned only when **src_dtype** and **C_dtype** are both unsigned.

4.12.5 tpu_bdc_int_sub

Perform subtraction of the src0 by src1 tensors, then perform arithmetic shift and saturation (optional) on the result.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (\text{src0}(n, c, h, w) - \text{src1}(n, c, h, w)) \text{ LSH } \text{shift} & \text{if shift} > 0 \\ (\text{src0}(n, c, h, w) - \text{src1}(n, c, h, w)) \text{ RSH } - \text{shift} & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src0_addr** – Address of the src0 tensor.
- **src1_addr** – Address of the src1 tensor.
- **shape** – Pointer to the shape of the destination, src0 and src1 tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src0_stride** – Pointer to the stride of the src0 tensor.

- **src1_stride** – Pointer to the stride of the src1 tensor.
- **dst_dtype** – Data type of the destination tensor.
- **src0_dtype** – Data type of the src0 tensor.
- **src1_dtype** – Data type of the src1 tensor.
- **shift** – Number of shifts.
- **rounding_mode** – Rounding mode for right shift.
- **saturation** – Flag of saturation.

Remarks

- The destination, src0 and src1 tensors start at the same NPU.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The **shift** is in [-32, 31].
- The valid choices of **src0_dtype** and **src1_dtype** are *DT_INT32*, *DT_UINT32*, *DT_INT16*, *DT_UINT16*, *DT_INT8* and *DT_UINT8*. The valid choices of **dst_dtype** are *DT_INT32*, *DT_INT16* and *DT_INT8*.

4.12.6 tpu_bdc_int_sub_C

Perform subtraction of the source tensor by a constant value, then perform arithmetic shift and saturation (optional) on the result.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (\text{src}(n, c, h, w) - C) \text{ LSH } \text{shift} & \text{if shift} > 0 \\ (\text{src}(n, c, h, w) - C) \text{ RSH } - \text{shift} & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **C** – Constant value.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dst_dtype** – Data type of the destination tensor.
- **src_dtype** – Data type of source tensor.
- **C_dtype** – Data type of *C*.
- **shift** – Number of shifts.
- **rounding_mode** – Rounding mode for right shift.
- **saturation** – Flag of saturation.

Remarks

- The destination and source tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The `shift` is in [-32, 31].
- The valid choices of `src_dtype` and `C_dtype` are `DT_INT32`, `DT_UINT32`, `DT_INT16`, `DT_UINT16`, `DT_INT8` and `DT_UINT8`. The valid choices of `dst_dtype` are `DT_INT32`, `DT_INT16` and `DT_INT8`.

4.12.7 tpu_bdc_int_C_sub

Perform subtraction of a constant value by the source tensor, then perform arithmetic shift and saturation (optional) on the result.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (C - \text{src}(n, c, h, w)) \text{ LSH } \text{shift} & \text{if shift} > 0 \\ (C - \text{src}(n, c, h, w)) \text{ RSH } - \text{shift} & \text{Otherwise.} \end{cases}$$

Parameters

- `dst_addr` – Address of the destination tensor.
- `src_addr` – Address of the source tensor.
- `C` – Constant value.
- `shape` – Pointer to the shape of the destination and source tensors.
- `dst_stride` – Pointer to the stride of the destination tensor.
- `src_stride` – Pointer to the stride of the source tensor.
- `dst_dtype` – Data type of the destination tensor.
- `src_dtype` – Data type of source tensor.
- `C_dtype` – Data type of `C`.
- `shift` – Number of shifts.
- `rounding_mode` – Rounding mode for right shift.
- `saturation` – Flag of saturation.

Remarks

- The destination and source tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The `shift` is in [-32, 31].
- The valid choices of `src_dtype` and `C_dtype` are `DT_INT32`, `DT_UINT32`, `DT_INT16`, `DT_UINT16`, `DT_INT8` and `DT_UINT8`. The valid choices of `dst_dtype` are `DT_INT32`, `DT_INT16` and `DT_INT8`.

4.12.8 tpu_bdc_int_pcs_sub

Perform subtraction of the src0 and src1 tensors, then perform arithmetic shift by channel and saturation (optional) on the result.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (\text{src0}(n, c, h, w) - \text{src1}(n, c, h, w)) \text{ LSH } \text{shift}(0, c, 0, 0) & \text{if } \text{shift}(0, c, 0, 0) > 0 \\ (\text{src0}(n, c, h, w) - \text{src1}(n, c, h, w)) \text{ RSH } - \text{shift}(0, c, 0, 0) & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src0_addr** – Address of the src0 tensor.
- **src1_addr** – Address of the src1 tensor.
- **shift_addr** – Address of the shift tensor.
- **shape** – Pointer to the shape of the destination, src0 and src1 tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src0_stride** – Pointer to the stride of the src0 tensor.
- **src1_stride** – Pointer to the stride of the src1 tensor.
- **dst_dtype** – Data type of the destination tensor.
- **src0_dtype** – Data type of the src0 tensor.
- **src1_dtype** – Data type of the src1 tensor.
- **rounding_mode** – Rounding mode for right shift.
- **saturation** – Flag of saturation.

Remarks

- The destination, src0, src1 and shift start at the same NPU.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The shape of shift is [1, **shape->c**, 1, 1], in compact layout. The data type of shift is **DT_INT8**, and the value range is [-32, 31].
- The valid choices of **src0_dtype** and **src1_dtype** are **DT_INT32**, **DT_UINT32**, **DT_INT16**, **DT_UINT16**, **DT_INT8** and **DT_UINT8**. The valid choices of **dst_dtype** are **DT_INT32**, **DT_INT16** and **DT_INT8**.

4.12.9 tpu_bdc_int_pcs_sub_C

Perform subtraction of the source tensor by a constant value, then perform arithmetic shift by channel and saturation (optional) on the result.

```
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_t *shape,
                          const dim4_t *dst_stride, const dim4_t *src_stride,
                          data_type_t dst_dtype, data_type_t src_dtype,
                          data_type_t C_dtype, rounding_mode_t rounding_mode,
                          bool saturation)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (\text{src}(n, c, h, w) - C) \text{ LSH } \text{shift}(0, c, 0, 0) & \text{if } \text{shift}(0, c, 0, 0) > 0 \\ (\text{src}(n, c, h, w) - C) \text{ RSH } - \text{shift}(0, c, 0, 0) & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **shift_addr** – Address of the shift tensor.
- **C** – Constant value.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dst_dtype** – Data type of the destination tensor.
- **src_dtype** – Data type of source tensor.
- **C_dtype** – Data type of *C*.
- **rounding_mode** – Rounding mode for right shift.
- **saturation** – Flag of saturation.

Remarks

- The destination, src and shift start at the same NPU.
- *shape->n*, *shape->c*, *shape->h* and *shape->w* are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The shape of shift is [1, *shape->c*, 1, 1], in compact layout. The data type of shift is *DT_INT8*, and the value range is [-32, 31].
- The valid choices of *src_dtype* and *C_dtype* are *DT_INT32*, *DT_UINT32*, *DT_INT16*, *DT_UINT16*, *DT_INT8* and *DT_UINT8*. The valid choices of *dst_dtype* are *DT_INT32*, *DT_INT16* and *DT_INT8*.

4.12.10 tpu_bdc_int_pcs_C_sub

Perform subtraction of a constant value by the source tensor, then perform arithmetic shift by channel and saturation (optional) on the result.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (C - \text{src}(n, c, h, w)) \text{ LSH } \text{shift}(0, c, 0, 0) & \text{if } \text{shift}(0, c, 0, 0) > 0 \\ (C - \text{src}(n, c, h, w)) \text{ RSH } - \text{shift}(0, c, 0, 0) & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **shift_addr** – Address of the shift tensor.
- **C** – Constant value.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dst_dtype** – Data type of the destination tensor.
- **src_dtype** – Data type of source tensor.
- **C_dtype** – Data type of *C*.
- **rounding_mode** – Rounding mode for right shift.
- **saturation** – Flag of saturation.

Remarks

- The destination, src and shift start at the same NPU.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The shape of shift is [1, **shape->c**, 1, 1], in compact layout. The data type of shift is *DT_INT8*, and the value range is [-32, 31].
- The valid choices of **src_dtype** and **C_dtype** are *DT_INT32*, *DT_UINT32*, *DT_INT16*, *DT_UINT16*, *DT_INT8* and *DT_UINT8*. The valid choices of **dst_dtype** are *DT_INT32*, *DT_INT16* and *DT_INT8*.

4.12.11 tpu_bdc_int_mul

Perform multiplication of the src0 and src1 tensors, then perform arithmetic shift and saturation (optional) on the result.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (\text{src0}(n, c, h, w) \times \text{src1}(n, c, h, w)) \text{ LSH } \text{shift} & \text{if } \text{shift} > 0 \\ (\text{src0}(n, c, h, w) \times \text{src1}(n, c, h, w)) \text{ RSH } - \text{shift} & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src0_addr** – Address of the src0 tensor.
- **src1_addr** – Address of the src1 tensor.
- **shape** – Pointer to the shape of the destination, src0 and src1 tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src0_stride** – Pointer to the stride of the src0 tensor.
- **src1_stride** – Pointer to the stride of the src1 tensor.
- **dst_dtype** – Data type of the destination tensor.
- **src0_dtype** – Data type of the src0 tensor.
- **src1_dtype** – Data type of the src1 tensor.
- **shift** – Number of shifts.
- **rounding_mode** – Rounding mode for right shift.
- **saturation** – Flag of saturation.

Remarks

- The destination, src0 and src1 tensors start at the same NPU.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- **shift** is in [-64, 31].
- The valid choices of **dst_dtype**, **src0_dtype** and **src1_dtype** are *DT_INT32*, *DT_UINT32*, *DT_INT16*, *DT_UINT16*, *DT_INT8* and *DT_UINT8*. **dst_dtype** is unsigned only when **src0_dtype** and **src1_dtype** are both unsigned.

4.12.12 tpu_bdc_int_mul_C

Perform multiplication of the source tensor and a constant value, then perform arithmetic shift and saturation (optional) on the result.

```
void tpu_bdc_int_mul_C(local_addr_t dst_addr, local_addr_t src_addr, scalar_t C,
                     const dim4_t *shape, const dim4_t *dst_stride, const dim4_t
                     *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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (\text{src}(n, c, h, w) \times C) \text{ LSH } \text{shift} & \text{if shift} > 0 \\ (\text{src}(n, c, h, w) \times C) \text{ RSH } -\text{shift} & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **C** – Constant value.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.

- **dst_dtype** – Data type of the destination tensor.
- **src_dtype** – Data type of source tensor.
- **C_dtype** – Data type of *C*.
- **shift** – Number of shifts.
- **rounding_mode** – Rounding mode for right shift.
- **saturation** – Flag of saturation.

Remarks

- The destination and source tensors start at the same NPU.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- **shift** is in [-64, 31].
- The valid choices of **dst_dtype**, **src_dtype** and **C_dtype** are *DT_INT32*, *DT_UINT32*, *DT_INT16*, *DT_UINT16*, *DT_INT8* and *DT_UINT8*. **dst_dtype** is unsigned only when **src_dtype** and **C_dtype** are both unsigned.

4.12.13 tpu_bdc_int_pcs_mul

Perform multiplication of the src0 and src1 tensors, then perform arithmetic shift by channel and saturation (optional) on the result.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (\text{src0}(n, c, h, w) \times \text{src1}(n, c, h, w)) \text{ LSH } \text{shift}(0, c, 0, 0) & \text{if } \text{shift}(0, c, 0, 0) > 0 \\ (\text{src0}(n, c, h, w) \times \text{src1}(n, c, h, w)) \text{ RSH } - \text{shift}(0, c, 0, 0) & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src0_addr** – Address of the src0 tensor.
- **src1_addr** – Address of the src1 tensor.
- **shift_addr** – Address of the shift tensor.
- **shape** – Pointer to the shape of the destination, src0 and src1 tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src0_stride** – Pointer to the stride of the src0 tensor.
- **src1_stride** – Pointer to the stride of the src1 tensor.
- **dst_dtype** – Data type of the destination tensor.
- **src0_dtype** – Data type of the src0 tensor.
- **src1_dtype** – Data type of the src1 tensor.
- **rounding_mode** – Rounding mode for right shift.
- **saturation** – Flag of saturation.

Remarks

- The destination, src0, src1 and shift start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The shape of shift is [1, `shape->c`, 1, 1], in compact layout. The data type of shift is `DT_INT8`, and the value range is [-64, 31].
- The valid choices of `dst_dtype`, `src0_dtype` and `src1_dtype` are `DT_INT32`, `DT_UINT32`, `DT_INT16`, `DT_UINT16`, `DT_INT8` and `DT_UINT8`. `dst_dtype` is unsigned only when `src0_dtype` and `src1_dtype` are both unsigned.

4.12.14 tpu_bdc_int_pcs_mul_C

Perform multiplication of the source tensor and a constant value, then perform arithmetic shift by channel and saturation (optional) on the result.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (\text{src}(n, c, h, w) \times C) \text{ LSH } \text{shift}(0, c, 0, 0) & \text{if } \text{shift}(0, c, 0, 0) > 0 \\ (\text{src}(n, c, h, w) \times C) \text{ RSH } - \text{shift}(0, c, 0, 0) & \text{Otherwise.} \end{cases}$$

Parameters

- `dst_addr` – Address of the destination tensor.
- `src_addr` – Address of the source tensor.
- `shift_addr` – Address of the shift tensor.
- `C` – Constant value.
- `shape` – Pointer to the shape of the destination and source tensors.
- `dst_stride` – Pointer to the stride of the destination tensor.
- `src_stride` – Pointer to the stride of the source tensor.
- `dst_dtype` – Data type of the destination tensor.
- `src_dtype` – Data type of source tensor.
- `C_dtype` – Data type of `C`.
- `rounding_mode` – Rounding mode for right shift.
- `saturation` – Flag of saturation.

Remarks

- The destination, src and shift start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The shape of shift is [1, `shape->c`, 1, 1], in compact layout. The data type of shift is `DT_INT8`, and the value range is [-64, 31].

- The valid choices of `dst_dtype`, `src_dtype` and `C_dtype` are `DT_INT32`, `DT_UINT32`, `DT_INT16`, `DT_UINT16`, `DT_INT8` and `DT_UINT8`. `dst_dtype` is unsigned only when `src_dtype` and `C_dtype` are both unsigned.

4.12.15 tpu_bdc_int8_mac

Perform multiplication of the `src0` and `src1` tensors, and accumulate results. Before accumulation, perform left arithmetic shift on the destination tensor. After accumulation, perform right arithmetic shift on the result.

```
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)
```

$$\text{dst}(n, c, h, w) = (\text{dst}(n, c, h, w) \text{ LSH } l\text{shift} + \text{src0}(n, c, h, w) \times \text{src1}(n, c, h, w)) \text{ RSH } r\text{shift}$$

Parameters

- `dst_addr` – Address of the destination tensor.
- `src0_addr` – Address of the `src0` tensor.
- `src1_addr` – Address of the `src1` tensor.
- `shape` – Pointer to the shape of the destination, `src0` and `src1` tensors.
- `dst_stride` – Pointer to the stride of the destination tensor.
- `src0_stride` – Pointer to the stride of the `src0` tensor.
- `src1_stride` – Pointer to the stride of the `src1` tensor.
- `src0_dtype` – Data type of the `src0` tensor.
- `src1_dtype` – Data type of the `src1` tensor.
- `lshift` – Number of left shifts.
- `rshift` – Number of right shifts.
- `rounding_mode` – Rounding mode for right shift.

Remarks

- The destination, `src0` and `src1` tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in `[1, 65535]`.
- If stride is `NULL`, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- `lshift` and `rshift` are both in `[0, 31]`.
- `src0_dtype` and `src1_dtype` are both `DT_INT8` or `DT_UINT8`. If `src0_dtype` and `src1_dtype` are both `DT_UINT8`, the data type of the destination tensor is `DT_UINT16`. The data type before accumulation is also recognized as `DT_UINT16`, otherwise it is `DT_INT16`.

4.12.16 tpu_bdc_int8_mac_C

Perform multiplication of the source tensor and a constant value, and accumulate results. Before accumulation, perform left arithmetic shift on the destination tensor. After accumulation, perform right arithmetic shift on the result.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **C** – Constant value.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **src_dtype** – Data type of the src0 tensor.
- **C_dtype** – Data type of *C*.
- **lshift** – Number of left shifts.
- **rshift** – Number of right shifts.
- **rounding_mode** – Rounding mode for right shift.

Remarks

- The destination and source tensors start at the same NPU.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- **lshift** and **rshift** are both in [0, 31].
- **src_dtype** and **C_dtype** are both *DT_INT8* or *DT_UINT8*. If **src_dtype** and **C_dtype** are both *DT_UINT8*, the data type of the destination tensor is *DT_UINT16*. The data type before accumulation is also recognized as *DT_UINT16*, otherwise it is *DT_INT16*.

4.12.17 tpu_bdc_int_min_C

Perform minimum operation of the source tensor and a constant value, then perform arithmetic shift on the result.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \min(\text{src}(n, c, h, w) + C) \text{ LSH } \text{shift} & \text{if } \text{shift} > 0 \\ \min(\text{src}(n, c, h, w) + C) \text{ RSH } -\text{shift} & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **C** – Constant value.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dtype** – Data type of destination, source tensors and **C**.
- **shift** – Number of shifts.
- **rounding_mode** – Rounding mode for right shift.

Remarks

- The destination and source tensors start at the same NPU.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535].
- If stride is NULL, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The **shift** is in [-32, 31].
- The valid choices of **dtype** is in *DT_INT32*, *DT_UINT32*, *DT_INT16*, *DT_UINT16*, *DT_INT8* and *DT_UINT8*.

4.12.18 tpu_bdc_int_max_C

Perform maximum operation of the source tensor and a constant value, then perform arithmetic shift on the result.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \max(\text{src}(n, c, h, w) + C) \text{ LSH } \text{shift} & \text{if } \text{shift} > 0 \\ \max(\text{src}(n, c, h, w) + C) \text{ RSH } - \text{shift} & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **C** – Constant value.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_stride** – Pointer to the stride of the destination tensor.
- **src_stride** – Pointer to the stride of the source tensor.
- **dtype** – Data type of destination, source tensors and **C**.
- **shift** – Number of shifts.
- **rounding_mode** – Rounding mode for right shift.

Remarks

- The destination and source tensors start at the same NPU.

- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in `[1, 65535]`.
- If `stride` is `NULL`, the relative tensor is in 64-byte aligned layout, otherwise in free layout.
- The `shift` is in `[-32, 31]`.
- The valid choices of `dtype` is in `DT_INT32`, `DT_UINT32`, `DT_INT16`, `DT_UINT16`, `DT_INT8` and `DT_UINT8`.

4.12.19 tpu_bdc_int_vc_add

Perform addition of the `src0` and `src1` vectors, then perform saturation (optional) on the result.

```
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)
```

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

Parameters

- `dst_addr` – Address of the destination matrix.
- `src0_addr` – Address of the `src0` vector.
- `src1_addr` – Address of the `src1` vector.
- `src0_len` – Length of `src0` vector.
- `src1_len` – Length of `src1` vector.
- `src0_len_per_channel` – Length of `src0` vector in each channel.
- `src1_len_per_channel` – Length of `src1` vector in each channel.
- `dst_dtype` – Data type of the destination matrix.
- `src0_dtype` – Data type of the `src0` vector.
- `src1_dtype` – Data type of the `src1` vector.
- `saturation` – Flag of saturation.

Remarks

- The destination matrix and `src1` vector start at the same NPU.
- `src0_len` and `src1_len` are in `[1, 65535]`.
- The destination matrix is in matrix layout, with rows of `src0_len` and columns of `src1_len`. The `src0` and `src1` vectors is in vector layout.
- The valid choices of `dst_dtype`, `src0_dtype` and `src1_dtype` are `DT_INT32`, `DT_UINT32`, `DT_INT16`, `DT_UINT16`, `DT_INT8` and `DT_UINT8`. `dst_dtype` is unsigned only when `src0_dtype` and `src1_dtype` are both unsigned.

4.12.20 tpu_bdc_int_vc_sub

Perform subtraction of the src0 and src1 vectors, then perform saturation (optional) on the result.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination matrix.
- **src0_addr** – Address of the src0 vector.
- **src1_addr** – Address of the src1 vector.
- **src0_len** – Length of src0 vector.
- **src1_len** – Length of src1 vector.
- **src0_len_per_channel** – Length of src0 vector in each channel.
- **src1_len_per_channel** – Length of src1 vector in each channel.
- **dst_dtype** – Data type of the destination matrix.
- **src0_dtype** – Data type of the src0 vector.
- **src1_dtype** – Data type of the src1 vector.
- **saturation** – Flag of saturation.

Remarks

- The destination matrix and src1 vector start at the same NPU.
- src0_len and src1_len are in [1, 65535].
- The destination matrix is in matrix layout, with rows of src0_len and columns of src1_len. The src0 and src1 vectors is in vector layout.
- The valid choices of dst_dtype, src0_dtype and src1_dtype are *DT_INT32*, *DT_UINT32*, *DT_INT16*, *DT_UINT16*, *DT_INT8* and *DT_UINT8*. The valid choices of dst_dtype are *DT_INT32*, *DT_INT16* and *DT_INT8*.

4.12.21 tpu_bdc_int_vc_mul

Perform multiplication of the src0 and src1 vectors, then perform saturation (optional) on the result.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination matrix.
- **src0_addr** – Address of the src0 vector.

- **src1_addr** – Address of the src1 vector.
- **src0_len** – Length of src0 vector.
- **src1_len** – Length of src1 vector.
- **src0_len_per_channel** – Length of src0 vector in each channel.
- **src1_len_per_channel** – Length of src1 vector in each channel.
- **dst_dtype** – Data type of the destination matrix.
- **src0_dtype** – Data type of the src0 vector.
- **src1_dtype** – Data type of the src1 vector.
- **saturation** – Flag of saturation.

Remarks

- The destination matrix and src1 vector start at the same NPU.
- **src0_len** and **src1_len** are in [1, 65535].
- The destination matrix is in matrix layout, with rows of **src0_len** and columns of **src1_len**. The src0 and src1 vectors is in vector layout.
- The valid choices of **dst_dtype**, **src0_dtype** and **src1_dtype** are [DT_INT32](#), [DT_UINT32](#), [DT_INT16](#), [DT_UINT16](#), [DT_INT8](#) and [DT_UINT8](#). **dst_dtype** is unsigned only when **src0_dtype** and **src1_dtype** are both unsigned.

4.13 Compare and Select Functions

4.13.1 tpu_bdc_greater_select

Compare whether the src0 is greater than the src1, and select the elements from the src2 or src3 tensors as the result.

```
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_t *shape, data_type_t src0_src1_dtype,
                           data_type_t dst_dtype)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{src2} & \text{if } \text{src0} > \text{src1} \\ \text{src3} & \text{Otherwise.} \end{cases}$$

For **src0**, **src1**, **src2** and **src3** in the above formula, if **src->type** is [TENSOR](#), get **src(n, c, h, w)**. If **src->type** is [VECTOR](#), get **src(0, c mod NPU_NUM, 0, 0)**. If **src->type** is [SCALAR](#), get the constant value.

Parameters

- **dst_addr** – Address of the destination tensor.
- **src0** – Address of the src0 tensor.Address of the src0 tensor.
- **src1** – Address of the src1 tensor.Address of the src1 tensor.
- **src2** – Address of the src2 tensor.Address of the src2 tensor.
- **src3** – Address of the src3 tensor.Address of the src3 tensor.
- **shape** – Pointer to the shape of the destination tensor.
- **src0_src1_dtype** – Data type of the src0 and src1 tensors.
- **dst_dtype** – Data type of the destination, src2 and src3 tensors.

Remarks

- The destination and source tensors start at the same NPU. The destination tensor is in 64-byte aligned layout.
- If `src->type` is *TENSOR*, the src and dst have the same shape, and are both in 64-byte aligned layout. If `src->type` is *VECTOR*, the shape of src is [1, `shape->c`, 1, 1].
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- The bit width of `src0_src1_dtype` is less than or equal to that of `dst_dtype`.

4.13.2 tpu_bdc_less_select

Compare whether the `src0` is less than the `src1`, and select the elements from the `src2` or `src3` tensors as the result.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{src2} & \text{if } \text{src0} < \text{src1} \\ \text{src3} & \text{Otherwise.} \end{cases}$$

For `src0`, `src1`, `src2` and `src3` in the above formula, if `src->type` is *TENSOR*, get `src(n, c, h, w)`. If `src->type` is *VECTOR*, get `src(0, c mod NPU_NUM, 0, 0)`. If `src->type` is *SCALAR*, get the constant value.

Parameters

- `dst_addr` – Address of the destination tensor.
- `src0` – Address of the `src0` tensor.
- `src1` – Address of the `src1` tensor.
- `src2` – Address of the `src2` tensor.
- `src3` – Address of the `src3` tensor.
- `shape` – Pointer to the shape of the destination tensor.
- `src0_src1_dtype` – Data type of the `src0` and `src1` tensors.
- `dst_dtype` – Data type of the destination, `src2` and `src3` tensors.

Remarks

- The destination and source tensors start at the same NPU. The destination tensor is in 64-byte aligned layout.
- If `src->type` is *TENSOR*, the src and dst have the same shape, and are both in 64-byte aligned layout. If `src->type` is *VECTOR*, the shape of src is [1, `shape->c`, 1, 1].
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- The bit width of `src0_src1_dtype` is less than or equal to that of `dst_dtype`.

4.13.3 tpu_bdc_equal_select

Compare whether the src0 is equal to the src1, and select the elements from the src2 or src3 tensors as the result.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{src2} & \text{if } \text{src0} = \text{src1} \\ \text{src3} & \text{Otherwise.} \end{cases}$$

For src0, src1, src2 and src3 in the above formula, if `src->type` is *TENSOR*, get `src(n, c, h, w)`. If `src->type` is *VECTOR*, get `src(0, c mod NPU_NUM, 0, 0)`. If `src->type` is *SCALAR*, get the constant value.

Parameters

- **dst_addr** – Address of the destination tensor.
- **src0** – Address of the src0 tensor.
- **src1** – Address of the src1 tensor.
- **src2** – Address of the src2 tensor.
- **src3** – Address of the src3 tensor.
- **shape** – Pointer to the shape of the destination tensor.
- **src0_src1_dtype** – Data type of the src0 and src1 tensors.
- **dst_dtype** – Data type of the destination, src2 and src3 tensors.

Remarks

- The destination and source tensors start at the same NPU. The destination tensor is in 64-byte aligned layout.
- If `src->type` is *TENSOR*, the src and dst have the same shape, and are both in 64-byte aligned layout. If `src->type` is *VECTOR*, the shape of src is [1, `shape->c`, 1, 1].
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in [1, 65535].
- The bit width of `src0_src1_dtype` is less than or equal to that of `dst_dtype`.

4.13.4 tpu_bdc_maximum_greater_select

Compare whether the src0 is greater than the src1. Select the maximum elements of src0 and src1 tensors as the dst0, and select the elements from the src2 or src3 tensors as the dst1.

```
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)
```

$$\text{dst0}(n, c, h, w) = \max(\text{src0}, \text{src1})$$

$$\text{dst1}(n, c, h, w) = \begin{cases} \text{src2} & \text{if } \text{src0} > \text{src1} \\ \text{src3} & \text{Otherwise.} \end{cases}$$

For `src0`, `src1`, `src2` and `src3` in the above formula, if `src->type` is *TENSOR*, get `src(n, c, h, w)`. If `src->type` is *VECTOR*, get `src(0, c mod NPU_NUM, 0, 0)`. If `src->type` is *SCALAR*, get the constant value.

Parameters

- **dst0_addr** – Address of the dst0 tensor.
- **dst1_addr** – Address of the dst1 tensor.
- **src0** – Address of the src0 tensor.
- **src1** – Address of the src1 tensor.
- **src2** – Address of the src2 tensor.
- **src3** – Address of the src3 tensor.
- **shape** – Pointer to the shape of the destination, src0 and src1 tensors.
- **dst0_dtype** – Data type of the src0 and src1 tensors.
- **dst1_dtype** – Data type of the dst1, src2 and src3 tensors.

Remarks

- The destination and source tensors start at the same NPU. The destination tensor is in 64-byte aligned layout.
- If `src->type` is *TENSOR*, the src and dst have the same shape, and are both in 64-byte aligned layout. If `src->type` is *VECTOR*, the shape of src is `[1, shape->c, 1, 1]`.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in `[1, 65535]`.
- The bit width of `dst0_dtype` is equal to that of `dst1_dtype`.

4.13.5 tpu_bdc_minimum_less_select

Compare whether the `src0` is less than the `src1`. Select the minimum elements of `src0` and `src1` tensors as the `dst0`, and select the elements from the `src2` or `src3` tensors as the `dst1`.

```
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)
```

$$\text{dst0}(n, c, h, w) = \min(\text{src0}, \text{src1})$$

$$\text{dst1}(n, c, h, w) = \begin{cases} \text{src2} & \text{if } \text{src0} < \text{src1} \\ \text{src3} & \text{Otherwise.} \end{cases}$$

For `src0`, `src1`, `src2` and `src3` in the above formula, if `src->type` is *TENSOR*, get `src(n, c, h, w)`. If `src->type` is *VECTOR*, get `src(0, c mod NPU_NUM, 0, 0)`. If `src->type` is *SCALAR*, get the constant value.

Parameters

- **dst0_addr** – Address of the dst0 tensor.
- **dst1_addr** – Address of the dst1 tensor.
- **src0** – Address of the src0 tensor.
- **src1** – Address of the src1 tensor.
- **src2** – Address of the src2 tensor.

- **src3** – Address of the src3 tensor.
- **shape** – Pointer to the shape of the destination, src0 and src1 tensors.
- **dst0_dtype** – Data type of the src0 and src1 tensors.
- **dst1_dtype** – Data type of the dst1, src2 and src3 tensors.

Remarks

- The destination and source tensors start at the same NPU. The destination tensor is in 64-byte aligned layout.
- If **src->type** is *TENSOR*, the src and dst have the same shape, and are both in 64-byte aligned layout. If **src->type** is *VECTOR*, the shape of src is [1, **shape->c**, 1, 1]. If **src->type** is *TENSOR*, the src and dst have the same shape, and are both in 64-byte aligned layout. If **src->type** is *VECTOR*, the shape of src is [1, **shape->c**, 1, 1].
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535].
- The bit width of **dst0_dtype** is equal to that of **dst1_dtype**.

4.14 Floating Point Matrix Functions

4.14.1 tpu_bdc_fp32_mm

Perform matrix multiplication with adding bias by column (optional), and perform result accumulation (optional).

```
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)
```

$$\text{output}_{M \times N} = \text{output}_{M \times N} + \text{left}_{M \times K} \times \text{right}_{K \times N} + \text{bias}_{1 \times N}$$

$$\text{output}(m, n) = \text{output}(m, n) + \left(\sum_k \text{left}(m, k) \times \text{right}(k, n) \right) + \text{bias}(0, n)$$

Parameters

- **output_addr** – Address of the output matrix.
- **left_addr** – Address of the left matrix.
- **right_addr** – Address of the right matrix.
- **bias_addr** – Address of the bias.
- **left_rows** – The number of rows of the left matrix.
- **left_cols** – The number of columns of the left matrix.
- **right_cols** – The number of columns of the right matrix.
- **left_cols_per_channel** – The number of columns in each channel of the left matrix.
- **right_cols_per_channel** – The number of columns in each channel of the right matrix.
- **has_bias** – Flag of has_bias.
- **result_add** – Flag of result_add.

Remarks

- The output, right and bias tensors start at the same NPU.
- `left_rows`, `left_cols` and `right_cols` are in [1, 65535].
- The output, left, right and bias are in matrix layout. The output matrix has the rows of `left_rows` and the columns of `right_cols`. The right matrix has the rows of `left_cols`. The bias has row of 1 and columns of `right_cols`.

4.14.2 tpu_bdc_fp32_mm_L_trans

Perform multiplication of the transposed left matrix and the right matrix, with adding bias by column (optional), and perform result accumulation (optional).

```
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)
```

$$\text{output}_{M \times N} = \text{output}_{M \times N} + \text{left}_{K \times M}^T \times \text{right}_{K \times N} + \text{bias}_{1 \times N}$$

$$\text{output}(m, n) = \text{output}(m, n) + \left(\sum_k \text{left}(k, m) \times \text{right}(k, n) \right) + \text{bias}(0, n)$$

Parameters

- **output_addr** – Address of the output matrix.
- **left_addr** – Address of the left matrix.
- **right_addr** – Address of the right matrix.
- **bias_addr** – Address of the bias.
- **left_rows** – The number of rows of the left matrix.
- **left_cols** – The number of columns of the left matrix.
- **right_cols** – The number of columns of the right matrix.
- **left_cols_per_channel** – The number of columns in each channel of the left matrix.
- **right_cols_per_channel** – The number of columns in each channel of the right matrix.
- **has_bias** – Flag of has_bias.
- **result_add** – Flag of result_add.

Remarks

- The output, right and bias tensors start at the same NPU.
- `left_rows`, `left_cols` and `right_cols` are in [1, 65535].
- The output, left, right and bias are in matrix layout. The output matrix has the rows of `left_cols` and the columns of `right_cols`. The right matrix has the rows of `left_rows`. The bias has row of 1 and columns of `right_cols`.

4.14.3 tpu_bdc_fp32_mm_L_const

Perform matrix multiplication with adding bias by column (optional), and perform result accumulation (optional). All elements of the left matrix have the same value.

```
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)
```

$$\text{output}_{M \times N} = \text{output}_{M \times N} + \text{left}_{M \times K} \times \text{right}_{K \times N} + \text{bias}_{1 \times N}$$

$$\text{output}(m, n) = \text{output}(m, n) + \left(\sum_k C \times \text{right}(k, n) \right) + \text{bias}(0, n)$$

Parameters

- **output_addr** – Address of the output matrix.
- **right_addr** – Address of the right matrix.
- **bias_addr** – Address of the bias.
- **C** – The constant value of left matrix.
- **left_rows** – The number of rows of the left matrix.
- **left_cols** – The number of columns of the left matrix.
- **right_cols** – The number of columns of the right matrix.
- **right_cols_per_channel** – The number of columns in each channel of the right matrix.
- **has_bias** – Flag of has_bias.
- **result_add** – Flag of result_add.

Remarks

- The output, right and bias tensors start at the same NPU.
- **left_rows**, **left_cols** and **right_cols** are in [1, 65535].
- The output, left, right and bias are in matrix layout. The output matrix has the rows of **left_rows** and the columns of **right_cols**. The right matrix has the rows of **left_cols**. The bias has row of 1 and columns of **right_cols**.

4.14.4 tpu_bdc_fp_mm

Perform matrix multiplication with result accumulation (optional).

```
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)
```

$$\text{output}_{M \times N} = \text{output}_{M \times N} + \text{left}_{M \times K} \times \text{right}_{K \times N}$$

$$\text{output}(m, n) = \text{output}(m, n) + \sum_k \text{left}(m, k) \times \text{right}(k, n)$$

Parameters

- **output_addr** – Address of the output matrix.
- **left_addr** – Address of the left matrix.

- **right_addr** – Address of the right matrix.
- **left_rows** – The number of rows of the left matrix.
- **left_cols** – The number of columns of the left matrix.
- **right_cols** – The number of columns of the right matrix.
- **output_dtype** – Data type of the output matrix.
- **left_right_dtype** – Data type of the left and right matrix.
- **result_add** – Flag of result_add.

Remarks

- The output, left and right start at NPU 0, and all in 64-byte aligned layout.
- The shape of output is [1, left_rows, 1, right_cols]. The shape of left is [1, left_rows, 1, left_cols]. The shape of right is [1, left_cols, 1, right_cols].
- left_rows, left_cols and right_cols are in [1, 65535].
- The valid choices of output_dtype are output_dtype is *DT_FP32*, or the same as left_right_dtype. The valid choices of left_right_dtype is *DT_FP16* or *DT_BFP16*.
- If perform accumulation on result, the data type of output before accumulation will be recognized as *DT_FP32*, not *DT_FP16* or *DT_FP16*.

4.14.5 tpu_bdc_fp_mm_R_trans

Perform multiplication of the left matrix and the transposed right matrix.

```
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)
```

$$\text{output}_{M \times N} = \text{left}_{M \times K} \times \text{right}_{N \times K}^T$$

$$\text{output}(m, n) = \sum_k \text{left}(m, k) \times \text{right}(n, k)$$

Parameters

- **output_addr** – Address of the output matrix.
- **left_addr** – Address of the left matrix.
- **right_addr** – Address of the right matrix.
- **left_rows** – The number of rows of the left matrix.
- **left_cols** – The number of columns of the left matrix.
- **right_rows** – The number of rows of the right matrix.
- **output_dtype** – Data type of the output matrix.
- **left_right_dtype** – Data type of the left and right matrix.

Remarks

- The output, left and right start at NPU 0, and all in 64-byte aligned layout.
- The shape of output is [1, left_rows, 1, right_rows]. The shape of left is [1, left_rows, 1, left_cols]. The shape of right is [1, right_rows, 1, left_cols].
- left_rows, left_cols and right_rows are in [1, 65535].

- The valid choices of `output_dtype` is `DT_FP32`, or the same as `left_right_dtype`. The valid choices of `left_right_dtype` is `DT_FP16` or `DT_BFP16`.

4.14.6 tpu_bdc_fp_mm_all_trans

Perform multiplication of the transposed left matrix and the transposed right matrix. Then perform result transposition and accumulation (optional).

```
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)
```

$$\text{output}_{N \times M}^T = \text{output}_{N \times M}^T + \text{left}_{K \times M}^T \times \text{right}_{N \times K}^T$$

$$\text{output}(n, m) = \text{output}(n, m) + \sum_k \text{left}(k, m) \times \text{right}(n, k)$$

Parameters

- `output_addr` – Address of the output matrix.
- `left_addr` – Address of the left matrix.
- `right_addr` – Address of the right matrix.
- `left_rows` – The number of rows of the left matrix.
- `left_cols` – The number of columns of the left matrix.
- `right_rows` – The number of rows of the right matrix.
- `output_dtype` – Data type of the output matrix.
- `left_right_dtype` – Data type of the left and right matrix.
- `result_add` – Flag of result_add.

Remarks

- The output, left and right start at NPU 0, and all in 64-byte aligned layout.
- The shape of output is `[1, right_rows, 1, left_cols]`. The shape of left is `[1, left_rows, 1, left_cols]`. The shape of right is `[1, right_rows, 1, left_rows]`.
- `left_rows`, `left_cols` and `right_rows` are in `[1, 65535]`.
- The valid choices of `output_dtype` is `DT_FP32`, or the same as `left_right_dtype`. The valid choices of `left_right_dtype` is `DT_FP16` or `DT_BFP16`.
- If perform accumulation on result, the data type of output before accumulation will be recognized as `DT_FP32`, not `DT_FP16` or `DT_FP16`.

4.14.7 tpu_bdc_fp_mm_L_const

Perform matrix multiplication and perform result accumulation (optional). All elements of the left matrix have the same value.

```
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)
```

$$\text{output}_{M \times N} = \text{output}_{M \times N} + \text{left}_{M \times K} \times \text{right}_{K \times N}$$

$$\text{output}(m, n) = \text{output}(m, n) + \sum_k C \times \text{right}(k, n)$$

Parameters

- **output_addr** – Address of the output matrix.
- **right_addr** – Address of the right matrix.
- **C** – The constant value of left matrix.
- **left_rows** – The number of rows of the left matrix.
- **left_cols** – The number of columns of the left matrix.
- **right_cols** – The number of columns of the right matrix.
- **output_dtype** – Data type of the output matrix.
- **right_C_dtype** – Data type of the right matrix and **C**.
- **result_add** – Flag of result_add.

Remarks

- The output and right start at the NPU 0, and both in 64-byte aligned layout.
- The shape of output is [1, left_rows, 1, right_cols]. The shape of left is [1, left_rows, 1, left_cols]. The shape of right is [1, left_cols, 1, right_cols].
- left_rows, left_cols and right_cols are in [1, 65535].
- The valid choices of output_dtype is *DT_FP32*, or the same as right_C_dtype. The valid choices of right_C_dtype is *DT_FP16* or *DT_BFP16*.
- If perform accumulation on result, the data type of output before accumulation will be recognized as *DT_FP32*, not *DT_FP16* or *DT_FP16*.

4.14.8 tpu_bdc_fp_mm_R_const

Perform matrix multiplication and perform result accumulation (optional). All elements of the right matrix have the same value.

```
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)
```

$$\text{output}_{M \times N} = \text{output}_{M \times N} + \text{left}_{M \times K} \times \text{right}_{K \times N}$$

$$\text{output}(m, n) = \text{output}(m, n) + \sum_k \text{left}(m, k) \times C$$

Parameters

- **output_addr** – Address of the output matrix.
- **left_addr** – Address of the left matrix.
- **C** – The constant value of right matrix.
- **left_rows** – The number of rows of the left matrix.
- **left_cols** – The number of columns of the left matrix.
- **right_cols** – The number of columns of the right matrix.
- **output_dtype** – Data type of the output matrix.

- **left_C_dtype** – Data type of the left matrix and **C**. Data type of the left matrix and **C**.
- **result_add** – Flag of result_add.

Remarks

- The output and left start at the NPU 0, and both in 64-byte aligned layout.
- The shape of output is [1, left_rows, 1, right_cols]. The shape of left is [1, left_rows, 1, left_cols]. The shape of right is [1, left_cols, 1, right_cols].
- left_rows, left_cols and right_cols are in [1, 65535].
- The valid choices of output_dtype is **DT_FP32**, or the same as left_C_dtype. The valid choices of left_C_dtype is **DT_FP16** or **DT_BFP16**.
- If perform accumulation on result, the data type of output before accumulation will be recognized as **DT_FP32**, not **DT_FP16** or **DT_FP16**.

4.14.9 tpu_bdc_fp_mm_L_const_R_trans

Perform multiplication of the left matrix and the transposed right matrix. All elements of the left matrix have the same value.

```
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)
```

$$\text{output}_{M \times N} = \text{left}_{M \times K} \times \text{right}_{N \times K}^T$$

$$\text{output}(m, n) = \sum_k C \times \text{right}(n, k)$$

Parameters

- **output_addr** – Address of the output matrix.
- **right_addr** – Address of the right matrix.
- **C** – The constant value of left matrix.
- **left_rows** – The number of rows of the left matrix.
- **left_cols** – The number of columns of the left matrix.
- **right_rows** – The number of rows of the right matrix.
- **output_dtype** – Data type of the output matrix.
- **right_C_dtype** – Data type of the right matrix and **C**.

Remarks

- The output and right start at the NPU 0, and both in 64-byte aligned layout.
- The shape of output is [1, left_rows, 1, right_rows]. The shape of left is [1, left_rows, 1, left_cols]. The shape of right is [1, right_rows, 1, left_cols].
- left_rows, left_cols and right_rows are in [1, 65535].
- The valid choices of output_dtype is **DT_FP32**, or the same as right_C_dtype. The valid choices of right_C_dtype is **DT_FP16** or **DT_BFP16**.

4.14.10 tpu_bdc_fp_mm_L_const_all_trans

Perform multiplication of the transposed left matrix and the transposed right matrix. Then perform result transposition and accumulation (optional). All elements of the left matrix have the same value.

```
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)
```

$$\text{output}_{N \times M}^T = \text{output}_{N \times M}^T + \text{left}_{K \times M}^T \times \text{right}_{N \times K}^T$$

$$\text{output}(n, m) = \text{output}(n, m) + \sum_k C \times \text{right}(n, k)$$

Parameters

- **output_addr** – Address of the output matrix.
- **right_addr** – Address of the right matrix.
- **C** – The constant value of left matrix.
- **left_rows** – The number of rows of the left matrix.
- **left_cols** – The number of columns of the left matrix.
- **right_rows** – The number of rows of the right matrix.
- **output_dtype** – Data type of the output matrix.
- **right_C_dtype** – Data type of the right matrix and *C*.
- **result_add** – Flag of result_add.

Remarks

- The output and right start at the NPU 0, and both in 64-byte aligned layout.
- The shape of output is [1, right_rows, 1, left_cols]. The shape of left is [1, left_rows, 1, left_cols]. The shape of right is [1, right_rows, 1, left_rows].
- left_rows, left_cols and right_rows are in [1, 65535].
- The valid choices of output_dtype is *DT_FP32*, or the same as right_C_dtype. The valid choices of right_C_dtype is *DT_FP16* or *DT_BFP16*.
- If perform accumulation on result, the data type of output before accumulation will be recognized as *DT_FP32*, not *DT_FP16* or *DT_FP16*.

4.14.11 tpu_bdc_fp_mm_R_const_all_trans

Perform multiplication of the transposed left matrix and the transposed right matrix. Then perform result transposition and accumulation (optional). All elements of the right matrix have the same value.

```
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)
```

$$\text{output}_{N \times M}^T = \text{output}_{N \times M}^T + \text{left}_{K \times M}^T \times \text{right}_{N \times K}^T$$

$$\text{output}(n, m) = \text{output}(n, m) + \sum_k \text{left}(k, m) \times C$$

Parameters

- **output_addr** – Address of the output matrix.
- **left_addr** – Address of the left matrix.
- **C** – The constant value of right matrix. The constant value of right matrix.
- **left_rows** – The number of rows of the left matrix.
- **left_cols** – The number of columns of the left matrix.
- **right_rows** – The number of rows of the right matrix.
- **output_dtype** – Data type of the output matrix.
- **left_C_dtype** – Data type of the left matrix and **C**.
- **result_add** – Flag of result_add.

Remarks

- The output and left start at the NPU 0, and both in 64-byte aligned layout.
- The shape of output is [1, right_rows, 1, left_cols]. The shape of left is [1, left_rows, 1, left_cols]. The shape of right is [1, right_rows, 1, left_rows].
- left_rows, left_cols and right_rows are in [1, 65535].
- The valid choices of output_dtype is *DT_FP32*, or the same as left_C_dtype. The valid choices of left_C_dtype is *DT_FP16* or *DT_BFP16*.
- If perform accumulation on result, the data type of output before accumulation will be recognized as *DT_FP32*, not *DT_FP16* or *DT_FP16*.

4.15 Fixed Point Matrix Functions

4.15.1 tpu_bdc_int_mm

Perform matrix multiplication and perform arithmetic shift on the result.

```
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)
```

$$\text{output}_{M \times N} = (\text{left}_{M \times K} \times \text{right}_{K \times N}) \text{ RSH} - \text{shift}$$

$$\text{output}(m, n) = \left(\sum_k \text{left}(m, k) \times \text{right}(k, n) \right) \text{ RSH} - \text{shift}$$

Parameters

- **output_addr** – Address of the output matrix.
- **left_addr** – Address of the left matrix.
- **right_addr** – Address of the right matrix.
- **left_rows** – The number of rows of the left matrix.
- **left_cols** – The number of columns of the left matrix.
- **right_cols** – The number of columns of the right matrix.
- **left_cols_per_channel** – The number of columns in each channel of the left matrix.

- **right_cols_per_channel** – The number of columns in each channel of the right matrix.
- **left_dtype** – Data type of the left matrix.
- **right_dtype** – Data type of the right matrix.
- **shift** – Number of shifts.
- **rounding_mode** – Rounding mode for right shift.

Remarks

- The output and right matrixs start at the same NPU.
- **left_rows**, **left_cols** and **right_cols** are in [1, 65535].
- The output, left, right are in matrix layout. The output matrix has the rows of **left_rows** and the columns of **right_cols**. The right matrix has the rows of **left_cols**.
- **left_dtype** and **right_dtype** have the same bit width, and the valid choices are *DT_INT32*, *DT_UINT32*, *DT_INT16*, *DT_UINT16*, *DT_INT8* and *DT_UINT8*. If **left_dtype** and **right_dtype** are both unsigned, the data type of the output matrix is *DT_UINT32*, otherwise is *DT_INT32*.
- The **shift** is in [-32, 0]. Shifting only works when **left_dtype** and **right_dtype** are *DT_INT32* or *DT_UINT32*.

4.15.2 tpu_bdc_int_mm_L_trans

Perform multiplication of the transposed left matrix and the right matrix, and perform arithmetic shift on the result.

```
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)
```

$$\text{output}_{M \times N} = (\text{left}_{K \times M}^T \times \text{right}_{K \times N}) \text{ RSH} - \text{shift}$$

$$\text{output}(m, n) = \left(\sum_k \text{left}(k, m) \times \text{right}(k, n) \right) \text{ RSH} - \text{shift}$$

Parameters

- **output_addr** – Address of the output matrix.
- **left_addr** – Address of the left matrix.
- **right_addr** – Address of the right matrix.
- **left_rows** – The number of rows of the left matrix.
- **left_cols** – The number of columns of the left matrix.
- **right_cols** – The number of columns of the right matrix.
- **left_cols_per_channel** – The number of columns in each channel of the left matrix.
- **right_cols_per_channel** – The number of columns in each channel of the right matrix.
- **left_dtype** – Data type of the left matrix.
- **right_dtype** – Data type of the right matrix.

- **shift** – Number of shifts.
- **rounding_mode** – Rounding mode for right shift.

Remarks

- The output and right matrixs start at the same NPU.
- **left_rows**, **left_cols** and **right_cols** are in [1, 65535].
- The output, left, right are in matrix layout. The output matrix has the rows of **left_cols** and the columns of **right_cols**. The right matrix has the rows of **left_rows**.
- **left_dtype** and **right_dtype** have the same bit width, and the valid choices are *DT_INT32*, *DT_UINT32*, *DT_INT16*, *DT_UINT16*, *DT_INT8* and *DT_UINT8*. If **left_dtype** and **right_dtype** are both unsigned, the data type of the output matrix is *DT_UINT32*, otherwise is *DT_INT32*.
- The **shift** is in [-32, 0]. Shifting only works when **left_dtype** and **right_dtype** are *DT_INT32* or *DT_UINT32*.

4.15.3 tpu_bdc_int_mm_L_const

Perform matrix multiplication and perform arithmetic shift on the result. All elements of the left matrix have the same value.

```
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)
```

$$\text{output}_{M \times N} = (\text{left}_{M \times K} \times \text{right}_{K \times N}) \text{ RSH } - \text{shift}$$

$$\text{output}(m, n) = \left(\sum_k C \times \text{right}(k, n) \right) \text{ RSH } - \text{shift}$$

Parameters

- **output_addr** – Address of the output matrix.
- **right_addr** – Address of the right matrix.
- **C** – Constant value.
- **left_rows** – The number of rows of the left matrix.
- **left_cols** – The number of columns of the left matrix.
- **right_cols** – The number of columns of the right matrix.
- **right_cols_per_channel** – The number of columns in each channel of the right matrix.
- **C_dtype** – Data type of *C*.
- **right_dtype** – Data type of the right matrix.
- **shift** – Number of shifts.
- **rounding_mode** – Rounding mode for right shift.

Remarks

- The output and right matrixs start at the same NPU.
- **left_rows**, **left_cols** and **right_cols** are in [1, 65535].

- The output and right are in matrix layout. The output matrix has the rows of `left_rows` and the columns of `right_cols`. The right matrix has the rows of `left_cols`.
- `C_dtype` and `right_dtype` have the same bit width, and the valid choices are `DT_INT32`, `DT_UINT32`, `DT_INT16`, `DT_UINT16`, `DT_INT8` and `DT_UINT8`. If `C_dtype` and `right_dtype` are both unsigned, the data type of the output matrix is `DT_UINT32`, otherwise is `DT_INT32`.
- The `shift` is in `[-32, 0]`. Shifting only works when `C_dtype` and `right_dtype` are `DT_INT32` or `DT_UINT32`.

4.15.4 tpu_bdc_int_pcs_mm

Perform matrix multiplication, and perform arithmetic shift on the result by column.

```
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)
```

$$\text{output}_{M \times N} = (\text{left}_{M \times K} \times \text{right}_{K \times N}) \text{ RSH } - \text{shift}_{1 \times N}$$

$$\text{output}(m, n) = \left(\sum_k \text{left}(m, k) \times \text{right}(k, n) \right) \text{ RSH } - \text{shift}(0, n)$$

Parameters

- **output_addr** – Address of the output matrix.
- **left_addr** – Address of the left matrix.
- **right_addr** – Address of the right matrix.
- **shift_addr** – Address of the shift.
- **left_rows** – The number of rows of the left matrix.
- **left_cols** – The number of columns of the left matrix.
- **right_cols** – The number of columns of the right matrix.
- **left_cols_per_channel** – The number of columns in each channel of the left matrix.
- **right_cols_per_channel** – The number of columns in each channel of the right matrix.
- **left_dtype** – Data type of the left matrix.
- **right_dtype** – Data type of the right matrix.
- **rounding_mode** – Rounding mode for right shift.

Remarks

- The output, right and shift start at the same NPU.
- `left_rows`, `left_cols` and `right_cols` are in `[1, 65535]`.
- The output, left, right and shift are in matrix layout. The output matrix has the rows of `left_rows` and the columns of `right_cols`. The right matrix has the rows of `left_cols`. The shift has the rows of 1 and the columns of `right_cols`.

- `left_dtype` and `right_dtype` have the same bit width, and the valid choices are `DT_INT32`, `DT_UINT32`, `DT_INT16`, `DT_UINT16`, `DT_INT8` and `DT_UINT8`. If `left_dtype` and `right_dtype` are both unsigned, the data type of the output matrix is `DT_UINT32`, otherwise is `DT_INT32`.
- The data type of the shift is `DT_INT8`, the value range is [-32, 0]. Shifting only works when `left_dtype` and `right_dtype` are `DT_INT32` or `DT_UINT32`.

4.15.5 tpu_bdc_int_pcs_mm_L_trans

Perform multiplication of the transposed left matrix and the right matrix, and perform arithmetic shift on the result by column.

```
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)
```

$$\text{output}_{M \times N} = (\text{left}_{K \times M}^T \times \text{right}_{K \times N}) \text{ RSH } - \text{shift}_{1 \times N}$$

$$\text{output}(m, n) = \left(\sum_k \text{left}(k, m) \times \text{right}(k, n) \right) \text{ RSH } - \text{shift}(0, n)$$

Parameters

- `output_addr` – Address of the output matrix.
- `left_addr` – Address of the left matrix.
- `right_addr` – Address of the right matrix.
- `shift_addr` – Address of the shift.
- `left_rows` – The number of rows of the left matrix.
- `left_cols` – The number of columns of the left matrix.
- `right_cols` – The number of columns of the right matrix.
- `left_cols_per_channel` – The number of columns in each channel of the left matrix.
- `right_cols_per_channel` – The number of columns in each channel of the right matrix.
- `left_dtype` – Data type of the left matrix.
- `right_dtype` – Data type of the right matrix.
- `rounding_mode` – Rounding mode for right shift.

Remarks

- The output, right and shift start at the same NPU.
- `left_rows`, `left_cols` and `right_cols` are in [1, 65535].
- The output, left, right and shift are in matrix layout. The output matrix has the rows of `left_cols` and the columns of `right_cols`. The right matrix has the rows of `left_rows`. The shift has the rows of 1 and the columns of `right_cols`.
- `left_dtype` and `right_dtype` have the same bit width, and the valid choices are `DT_INT32`, `DT_UINT32`, `DT_INT16`, `DT_UINT16`, `DT_INT8` and `DT_UINT8`. If `left_dtype` and `right_dtype` are both unsigned, the data type of the output matrix is `DT_UINT32`, otherwise is `DT_INT32`.

- The data type of the shift is `DT_INT8`, the value range is [-32, 0]. Shifting only works when `left_dtype` and `right_dtype` are `DT_INT32` or `DT_UINT32`.

4.15.6 tpu_bdc_int_pcs_mm_L_const

Perform matrix multiplication and perform arithmetic shift on the result by column. All elements of the left matrix have the same value.

```
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)
```

$$\text{output}_{M \times N} = (\text{left}_{M \times K} \times \text{right}_{K \times N}) \text{ RSH } - \text{shift}_{1 \times N}$$

$$\text{output}(m, n) = \left(\sum_k C \times \text{right}(k, n) \right) \text{ RSH } - \text{shift}(0, n)$$

Parameters

- `output_addr` – Address of the output matrix.
- `right_addr` – Address of the right matrix.
- `shift_addr` – Address of the shift.
- `C` – Constant value.
- `left_rows` – The number of rows of the left matrix.
- `left_cols` – The number of columns of the left matrix.
- `right_cols` – The number of columns of the right matrix.
- `right_cols_per_channel` – The number of columns in each channel of the right matrix.
- `C_dtype` – Data type of `C`.
- `right_dtype` – Data type of the right matrix.
- `rounding_mode` – Rounding mode for right shift.

Remarks

- The output, right and shift start at the same NPU.
- `left_rows`, `left_cols` and `right_cols` are in [1, 65535].
- The output, right and shift are in matrix layout. The output matrix has the rows of `left_rows` and the columns of `right_cols`. The right matrix has the rows of `left_cols`. The shift has the rows of 1 and the columns of `right_cols`.
- `C_dtype` and `right_dtype` have the same bit width, and the valid choices are `DT_INT32`, `DT_UINT32`, `DT_INT16`, `DT_UINT16`, `DT_INT8` and `DT_UINT8`. If `C_dtype` and `right_dtype` are both unsigned, the data type of the output matrix is `DT_UINT32`, otherwise is `DT_INT32`.
- The data type of the shift is `DT_INT8`, the value range is [-32, 0]. Shifting only works when `C_dtype` and `right_dtype` are `DT_INT32` or `DT_UINT32`.

4.15.7 tpu_bdc_int8_mm

Firstly, perform matrix multiplication with adding bias by column (optional) and perform left shift on the output before accumulation (optional). Then perform result accumulation (optional) and perform ReLU on the result after accumulation (optional). Finally perform right shift and saturation on the result.

```
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)
```

$$\text{output}_{M \times N} = \text{ReLU}((\text{output}_{M \times N} \text{ LSH } l\text{shift}) + \text{left}_{M \times K} \times \text{right}_{K \times N} + \text{bias}_{1 \times N}) \text{ RSH } r\text{shift}$$

$$\text{output}(m, n) = \text{ReLU}(((\text{output}(m, n) \text{ LSH } l\text{shift}) + (\sum_k \text{left}(m, k) \times \text{right}(k, n)) + \text{bias}(0, n))) \text{ RSH } r\text{shift}$$

Parameters

- **output_addr** – Address of the output matrix.
- **left_addr** – Address of the left matrix.
- **right_addr** – Address of the right matrix.
- **bias_addr** – Address of the bias.Address of the bias.
- **left_rows** – The number of rows of the left matrix.
- **left_cols** – The number of columns of the left matrix.
- **right_cols** – The number of columns of the right matrix.
- **left_cols_per_channel** – The number of columns in each channel of the left matrix.
- **right_cols_per_channel** – The number of columns in each channel of the right matrix.
- **output_dtype** – Data type of the output matrix.
- **left_dtype** – Data type of the left matrix.
- **right_dtype** – Data type of the right matrix.
- **bias_dtype** – Data type of the bias.
- **lshift** – Number of left shifts.
- **rshift** – Number of right shifts.
- **has_bias** – Flag of adding bias.
- **result_add** – Flag of result accumulation.
- **result_relu** – Flag of result ReLU.

Remarks

- The output, right and bias tensors start at the same NPU.
- **left_rows**, **left_cols** and **right_cols** are in [1, 65535].
- The output, left, right and bias are in matrix layout. The output matrix has the rows of **left_rows** and the columns of **right_cols**. The right matrix has the rows of **left_cols**.The bias has row of 1 and columns of **right_cols**.

- The valid choices of `output_dtype` are `DT_INT16`, `DT_UINT16`, `DT_INT8` and `DT_UINT8`. The valid choices of `left_dtype` and `right_dtype` is `DT_INT8` and `DT_UINT8`. The valid choices of `bias_dtype` are `DT_INT16` and `DT_UINT16`.
- If `left_dtype`, `right_dtype` and `bias_dtype` (if `has_bias`) are all unsigned, the data type of output matrix before accumulation is recognized as unsigned, otherwise as signed.
- If `left_dtype`, `right_dtype` and `bias_dtype` (if `has_bias`) are all unsigned, or `result_relu` is true, the data type of the output matrix is recognized as unsigned, otherwise as signed.
- If `output_dtype` is `DT_INT8` or `DT_UINT8` and perform accumulation on the result, the data type of the output matrix before the accumulation will be recognized as `DT_INT16` or `DT_UINT16`, not `DT_INT8` or `DT_UINT8`.
- The `lshift` and `rshift` is in `[0, 31]`.

4.15.8 tpu_bdc_int8_mm_L_trans

Firstly, perform multiplication of the transposed left matrix and the right matrix with adding bias by column (optional), and perform left shift on the output before accumulation (optional). Then perform result accumulation (optional) and perform ReLU on the result after accumulation (optional). Finally perform right shift and saturation on the result.

```
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)
```

$$\text{output}_{M \times N} = \text{ReLU}((\text{output}_{M \times N} \text{ LSH } lshift) + \text{left}_{K \times M}^T \times \text{right}_{K \times N} + \text{bias}_{1 \times N}) \text{ RSH } rshift$$

$$\text{output}(m, n) = \text{ReLU}(((\text{output}(m, n) \text{ LSH } lshift) + (\sum_k \text{left}(k, m) \times \text{right}(k, n)) + \text{bias}(0, n))) \text{ RSH } rshift$$

Parameters

- `output_addr` – Address of the output matrix.
- `left_addr` – Address of the left matrix.
- `right_addr` – Address of the right matrix.
- `bias_addr` – Address of the bias.
- `left_rows` – The number of rows of the left matrix.
- `left_cols` – The number of columns of the left matrix.
- `right_cols` – The number of columns of the right matrix.
- `left_cols_per_channel` – The number of columns in each channel of the left matrix.
- `right_cols_per_channel` – The number of columns in each channel of the right matrix.
- `output_dtype` – Data type of the output matrix.
- `left_dtype` – Data type of the left matrix.
- `right_dtype` – Data type of the right matrix.

- **bias_dtype** – Data type of the bias.
- **lshift** – Number of left shifts.
- **rshift** – Number of right shifts.
- **has_bias** – Flag of adding bias. Flag of adding bias.
- **result_add** – Flag of result accumulation.
- **result_relu** – Flag of result ReLU.

Remarks

- The output, right and bias tensors start at the same NPU.
- **left_rows**, **left_cols** and **right_cols** are in [1, 65535].
- The output, left, right and bias are in matrix layout. The output matrix has the rows of **left_cols** and the columns of **right_cols**. The right matrix has the rows of **left_rows**. The bias has row of 1 and columns of **right_cols**.
- The valid choices of **output_dtype** are *DT_INT16*, *DT_UINT16*, *DT_INT8* and *DT_UINT8*. The valid choices of **left_dtype** and **right_dtype** is *DT_INT8* and *DT_UINT8*. The valid choices of **bias_dtype** are *DT_INT16* and *DT_UINT16*.
- If **left_dtype**, **right_dtype** and **bias_dtype** (if **has_bias**) are all unsigned, the data type of output matrix before accumulation is recognized as unsigned, otherwise as signed.
- If **left_dtype**, **right_dtype** and **bias_dtype** (if **has_bias**) are all unsigned, or **result_relu** is true, the data type of the output matrix is recognized as unsigned, otherwise as signed.
- If **output_dtype** is *DT_INT8* or *DT_UINT8* and perform accumulation on the result, the data type of the output matrix before the accumulation will be recognized as *DT_INT16* or *DT_UINT16*, not *DT_INT8* or *DT_UINT8*.
- The **lshift** and **rshift** is in [0, 31].

4.15.9 tpu_bdc_int8_mm_L_const

Firstly, perform multiplication of the left matrix and the right matrix with adding bias by column (optional), and perform left shift on the output before accumulation (optional). Then perform result accumulation (optional) and perform ReLU on the result after accumulation (optional). Finally perform right shift and saturation on the result. All elements of the left matrix have the same value.

```
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)
```

$$\text{output}_{M \times N} = \text{ReLU}((\text{output}_{M \times N} \text{ LSH } lshift) + \text{left}_{M \times K} \times \text{right}_{K \times N} + \text{bias}_{1 \times N}) \text{ RSH } rshift$$

$$\text{output}(m, n) = \text{ReLU}(((\text{output}(m, n) \text{ LSH } lshift) + (\sum_k C \times \text{right}(k, n)) + \text{bias}(0, n))) \text{ RSH } rshift$$

Parameters

- **output_addr** – Address of the output matrix.
- **right_addr** – Address of the right matrix.
- **bias_addr** – Address of the bias.

- **C** – Constant value.
- **left_rows** – The number of rows of the left matrix.
- **left_cols** – The number of columns of the left matrix.
- **right_cols** – The number of columns of the right matrix.
- **right_cols_per_channel** – The number of columns in each channel of the right matrix.
- **output_dtype** – Data type of the output matrix.
- **C_dtype** – Data type of **C**.
- **right_dtype** – Data type of the right matrix.
- **bias_dtype** – Data type of the bias.
- **lshift** – Number of left shifts.
- **rshift** – Number of right shifts.
- **has_bias** – Flag of adding bias.
- **result_add** – Flag of result accumulation.
- **result_relu** – Flag of result ReLU.

Remarks

- The output, right and bias tensors start at the same NPU.
- **left_rows**, **left_cols** and **right_cols** are in [1, 65535].
- The output, right and bias are in matrix layout. The output matrix has the rows of **left_rows** and the columns of **right_cols**. The right matrix has the rows of **left_cols**. The bias has row of 1 and columns of **right_cols**.
- The valid choices of **output_dtype** are **DT_INT16**, **DT_UINT16**, **DT_INT8** and **DT_UINT8**. The valid choices of **C_dtype** and **right_dtype** are **DT_INT8** and **DT_UINT8**. The valid choices of **bias_dtype** are **DT_INT16** and **DT_UINT16**.
- If **C_dtype**, **right_dtype** and **bias_dtype** (if **has_bias**) are all unsigned, the data type of output matrix before accumulation is recognized as unsigned, otherwise as signed.
- If **C_dtype**, **right_dtype** and **bias_dtype** (if **has_bias**) are all unsigned, or **result_relu** is true, the data type of the output matrix is recognized as unsigned, otherwise as signed.
- If **output_dtype** is **DT_INT8** or **DT_UINT8** and perform accumulation on the result, the data type of the output matrix before the accumulation will be recognized as **DT_INT16** or **DT_UINT16**, not **DT_INT8** or **DT_UINT8**.
- The **lshift** and **rshift** is in [0, 31].

4.15.10 tpu_bdc_int8_zp_mm

Perform matrix multiplication and perform result accumulation (optional) without result saturation. All elements of the right matrix are subtracted by zero-point.

```
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)
```

$$\text{output}_{M \times N} = \text{output}_{M \times N} + \text{left}_{M \times K} \times (\text{right}_{K \times N} - \text{zp})$$

$$\text{output}(m, n) = \text{output}(m, n) + \sum_k \text{left}(m, k) \times (\text{right}(k, n) - \text{zp})$$

Parameters

- **output_addr** – Address of the output matrix.
- **left_addr** – Address of the left matrix.
- **right_addr** – Address of the right matrix.
- **zp_val** – Value of zero-point.
- **left_rows** – The number of rows of the left matrix.
- **left_cols** – The number of columns of the left matrix.
- **right_cols** – The number of columns of the right matrix.
- **left_dtype** – Data type of the left matrix.
- **right_zp_dtype** – Data type of the right matrix and *zp_val*.
- **result_add** – Flag of result accumulation.

Remarks

- The output, left and right start at the NPU 0, and all in 64-byte aligned layout.
- The shape of output is [1, left_rows, 1, right_cols]. The shape of left is [1, left_rows, 1, left_cols]. The shape of right is [1, left_cols, 1, right_cols].
- left_rows, left_cols and right_cols are in [1, 65535].
- Data type of the output matrix is *DT_INT32*. The valid choices of left_dtype and right_zp_dtype are *DT_INT8* and *DT_UINT8*.

4.15.11 tpu_bdc_int8_zp_mm_R_trans

Perform multiplication of the left matrix and the transposed right matrix without result saturation. All elements of the right matrix are subtracted by zero-point.

```
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)
```

$$\text{output}_{M \times N} = \text{left}_{M \times K} \times (\text{right}_{N \times K}^T - \text{zp})$$

$$\text{output}(m, n) = \sum_k \text{left}(m, k) \times (\text{right}(n, k) - \text{zp})$$

Parameters

- **output_addr** – Address of the output matrix.
- **left_addr** – Address of the left matrix.
- **right_addr** – Address of the right matrix.
- **zp_val** – Value of zero-point.
- **left_rows** – The number of rows of the left matrix.
- **left_cols** – The number of columns of the left matrix.
- **right_rows** – The number of rows of the right matrix.
- **left_dtype** – Data type of the left matrix.
- **right_zp_dtype** – Data type of the right matrix and *zp_val*.

Remarks

- The output, left and right start at the NPU 0, and all in 64-byte aligned layout.
- The shape of output is `[1, left_rows, 1, right_rows]`. The shape of left is `[1, left_rows, 1, left_cols]`. The shape of right is `[1, right_rows, 1, left_cols]`.
- `left_rows`, `left_cols` and `right_rows` are in `[1, 65535]`.
- Data type of the output matrix is `DT_INT32`. The valid choices of `left_dtype` and `right_zp_dtype` are `DT_INT8` and `DT_UINT8`.

4.15.12 tpu_bdc_int8_zp_mm_all_trans

Perform multiplication of the transposed left matrix and the transposed right matrix. All elements of the right matrix are subtracted by zero-point. Then perform result transposition and accumulation (optional), without result 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)
```

$$\text{output}_{N \times M}^T = \text{output}_{N \times M}^T + \text{left}_{K \times M}^T \times (\text{right}_{N \times K}^T - \text{zp})$$

$$\text{output}(n, m) = \text{output}(n, m) + \sum_k \text{left}(k, m) \times (\text{right}(n, k) - \text{zp})$$

Parameters

- `output_addr` – Address of the output matrix.
- `left_addr` – Address of the left matrix.
- `right_addr` – Address of the right matrix.
- `zp_val` – Value of zero-point.
- `left_rows` – The number of rows of the left matrix.
- `left_cols` – The number of columns of the left matrix.
- `right_rows` – The number of rows of the right matrix.
- `left_dtype` – Data type of the left matrix.
- `right_zp_dtype` – Data type of the right matrix and `zp_val`.
- `result_add` – Flag of result accumulation.

Remarks

- The output, left and right start at the NPU 0, and all in 64-byte aligned layout.
- The shape of output is `[1, right_rows, 1, left_cols]`. The shape of left is `[1, left_rows, 1, left_cols]`. The shape of right is `[1, right_rows, 1, left_rows]`.
- `left_rows`, `left_cols` and `right_rows` are in `[1, 65535]`.
- Data type of the output matrix is `DT_INT32`. The valid choices of `left_dtype` and `right_zp_dtype` are `DT_INT8` and `DT_UINT8`.

4.15.13 tpu_bdc_int8_zp_mm_L_const

Perform matrix multiplication and perform result accumulation (optional) without result saturation. All elements of the left matrix have the same value. All elements of the right matrix are subtracted by zero-point.

```
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)
```

$$\text{output}_{M \times N} = \text{output}_{M \times N} + \text{left}_{M \times K} \times (\text{right}_{K \times N} - \text{zp})$$

$$\text{output}(m, n) = \text{output}(m, n) + \sum_k C \times (\text{right}(k, n) - \text{zp})$$

Parameters

- **output_addr** – Address of the output matrix.
- **right_addr** – Address of the right matrix.
- **C** – The constant value of the left matrix.
- **zp_val** – Value of zero-point.
- **left_rows** – The number of rows of the left matrix.
- **left_cols** – The number of columns of the left matrix.
- **right_cols** – The number of columns of the right matrix.
- **C_dtype** – Data type of *C*.
- **right_zp_dtype** – Data type of the right matrix and *zp_val*.
- **result_add** – Flag of result accumulation.

Remarks

- The output and right start at the NPU 0, and all in 64-byte aligned layout.
- The shape of output is [1, left_rows, 1, right_cols]. The shape of left is [1, left_rows, 1, left_cols]. The shape of right is [1, left_cols, 1, right_cols].
- left_rows, left_cols and right_cols are in [1, 65535].
- Data type of the output matrix is *DT_INT32*. The valid choices of *C_dtype* and *right_zp_dtype* are *DT_INT8* and *DT_UINT8*.

4.15.14 tpu_bdc_int8_zp_mm_R_const

Perform matrix multiplication and perform result accumulation (optional) without result saturation. All elements of the right matrix have the same value and are subtracted by zero-point.

```
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)
```

$$\text{output}_{M \times N} = \text{output}_{M \times N} + \text{left}_{M \times K} \times (\text{right}_{K \times N} - \text{zp})$$

$$\text{output}(m, n) = \text{output}(m, n) + \sum_k \text{left}(m, k) \times (C - \text{zp})$$

Parameters

- **output_addr** – Address of the output matrix.
- **left_addr** – Address of the left matrix.
- **C** – The constant value of the right matrix.
- **zp_val** – Value of zero-point.
- **left_rows** – The number of rows of the left matrix.
- **left_cols** – The number of columns of the left matrix.
- **right_cols** – The number of columns of the right matrix.
- **left_dtype** – Data type of the left matrix.
- **C_zp_dtype** – Data type of *C* and *zp_val*.
- **result_add** – Flag of result accumulation.

Remarks

- The output and left start at the NPU 0, and all in 64-byte aligned layout.
- The shape of output is [1, left_rows, 1, right_cols]. The shape of left is [1, left_rows, 1, left_cols]. The shape of right is [1, left_cols, 1, right_cols].
- left_rows, left_cols and right_cols are in [1, 65535].
- Data type of the output matrix is *DT_INT32*. The valid choices of left_dtype and C_zp_dtype are *DT_INT8* and *DT_UINT8*.

4.15.15 tpu_bdc_int8_zp_mm_L_const_R_trans

Perform multiplication of the left matrix and the transposed right matrix without result saturation. All elements of the left matrix have the same value. All elements of the right matrix are subtracted by zero-point.

```
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)
```

$$\text{output}_{M \times N} = \text{left}_{M \times K} \times (\text{right}_{N \times K}^T - \text{zp})$$

$$\text{output}(m, n) = \sum_k C \times (\text{right}(n, k) - \text{zp})$$

Parameters

- **output_addr** – Address of the output matrix.
- **right_addr** – Address of the right matrix.
- **C** – The constant value of the left matrix.
- **zp_val** – Value of zero-point.
- **left_rows** – The number of rows of the left matrix.
- **left_cols** – The number of columns of the left matrix.
- **right_rows** – The number of rows of the right matrix.
- **C_dtype** – Data type of *C*.
- **right_zp_dtype** – Data type of the right matrix and *zp_val*.

Remarks

- The output and right start at the NPU 0, and all in 64-byte aligned layout.
- The shape of output is `[1, left_rows, 1, right_rows]`. The shape of left is `[1, left_rows, 1, left_cols]`. The shape of right is `[1, right_rows, 1, left_cols]`.
- `left_rows`, `left_cols` and `right_rows` are in `[1, 65535]`.
- Data type of the output matrix is `DT_INT32`. The valid choices of `C_dtype` and `right_zp_dtype` are `DT_INT8` and `DT_UINT8`.

4.15.16 tpu_bdc_int8_zp_mm_L_const_all_trans

Perform multiplication of the transposed left matrix and the transposed right matrix. Then perform result transposition and accumulation (optional) without result saturation. All elements of the left matrix have the same value. All elements of the right matrix are subtracted by zero-point.

```
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)
```

$$\text{output}_{N \times M}^T = \text{output}_{N \times M}^T + \text{left}_{K \times M}^T \times (\text{right}_{N \times K}^T - \text{zp})$$

$$\text{output}(n, m) = \text{output}(n, m) + \sum_k C \times (\text{right}(n, k) - \text{zp})$$

Parameters

- `output_addr` – Address of the output matrix.
- `right_addr` – Address of the right matrix.
- `C` – The constant value of the left matrix.
- `zp_val` – Value of zero-point.
- `left_rows` – The number of rows of the left matrix.
- `left_cols` – The number of columns of the left matrix.
- `right_rows` – The number of rows of the right matrix.
- `C_dtype` – Data type of `C`.
- `right_zp_dtype` – Data type of the right matrix and `zp_val`.
- `result_add` – Flag of result accumulation.

Remarks

- The output and right start at the NPU 0, and all in 64-byte aligned layout.
- The shape of output is `[1, right_rows, 1, left_cols]`. The shape of left is `[1, left_rows, 1, left_cols]`. The shape of right is `[1, right_rows, 1, left_rows]`.
- `left_rows`, `left_cols` and `right_rows` are in `[1, 65535]`.
- Data type of the output matrix is `DT_INT32`. The valid choices of `C_dtype` and `right_zp_dtype` are `DT_INT8` and `DT_UINT8`.

4.15.17 tpu_bdc_int8_zp_mm_R_const_all_trans

Perform multiplication of the transposed left matrix and the transposed right matrix. Then perform result transposition and accumulation (optional) without result saturation. All elements of the right matrix have the same value and are subtracted by zero-point.

```
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)
```

$$\text{output}_{N \times M}^T = \text{output}_{N \times M}^T + \text{left}_{K \times M}^T \times (\text{right}_{N \times K}^T - \text{zp})$$

$$\text{output}(n, m) = \text{output}(n, m) + \sum_k \text{left}(k, m) \times (C - \text{zp})$$

Parameters

- **output_addr** – Address of the output matrix.
- **left_addr** – Address of the left matrix.
- **C** – The constant value of the right matrix.
- **zp_val** – Value of zero-point.
- **left_rows** – The number of rows of the left matrix.
- **left_cols** – The number of columns of the left matrix.
- **right_rows** – The number of rows of the right matrix.
- **left_dtype** – Data type of the left matrix.
- **C_zp_dtype** – Data type of *C* and *zp_val*.
- **result_add** – Flag of result accumulation.

Remarks

- The output and left start at the NPU 0, and all in 64-byte aligned layout.
- The shape of output is [1, right_rows, 1, left_cols]. The shape of left is [1, left_rows, 1, left_cols]. The shape of right is [1, right_rows, 1, left_rows].
- left_rows, left_cols and right_rows are in [1, 65535].
- Data type of the output matrix is *DT_INT32*. The valid choices of left_dtype and C_zp_dtype are *DT_INT8* and *DT_UINT8*.

4.15.18 tpu_bdc_int8_pc_zp_mm

Perform matrix multiplication and perform result accumulation (optional) without result saturation. Subtract zero-point from the right matrix by column.

```
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)
```

$$\text{output}_{M \times N} = \text{output}_{M \times N} + \text{left}_{M \times K} \times (\text{right}_{K \times N} - \text{zp}_{1 \times N})$$

$$\text{output}(m, n) = \text{output}(m, n) + \sum_k \text{left}(m, k) \times (\text{right}(k, n) - \text{zp}(0, n))$$

Parameters

- **output_addr** – Address of the output matrix.
- **left_addr** – Address of the left matrix.
- **right_addr** – Address of the right matrix.
- **zp_addr** – Address of the zero-point.
- **left_rows** – The number of rows of the left matrix.
- **left_cols** – The number of columns of the left matrix.
- **right_cols** – The number of columns of the right matrix.
- **left_dtype** – Data type of the left matrix.
- **right_zp_dtype** – Data type of the right matrix and zero-point.
- **result_add** – Flag of result accumulation.

Remarks

- The output, left, right and zp start at the NPU 0, and all in 64-byte aligned layout.
- The shape of output is [1, left_rows, 1, right_cols]. The shape of left is [1, left_rows, 1, left_cols]. The shape of right is [1, left_cols, 1, right_cols]. The shape of zero-point is [1, *NPU_NUM*, 1, right_cols], not [1, 1, 1, right_cols]. Each NPU has the same one.
- left_rows, left_cols and right_cols are in [1, 65535].
- Data type of the output matrix is *DT_INT32*. The valid choices of left_dtype and right_zp_dtype are *DT_INT8* and *DT_UINT8*.

4.15.19 tpu_bdc_int8_pc_zp_mm_R_trans

Perform multiplication of the left matrix and the transposed right matrix without result saturation. Subtract zero-point from the transposed right matrix by column.

```
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)
```

$$\text{output}_{M \times N} = \text{left}_{M \times K} \times (\text{right}_{N \times K}^T - \text{zp}_{1 \times N})$$

$$\text{output}(m, n) = \sum_k \text{left}(m, k) \times (\text{right}(n, k) - \text{zp}(0, n))$$

Parameters

- **output_addr** – Address of the output matrix.
- **left_addr** – Address of the left matrix.
- **right_addr** – Address of the right matrix.
- **zp_addr** – Address of the zero-point.
- **left_rows** – The number of rows of the left matrix.
- **left_cols** – The number of columns of the left matrix.
- **right_rows** – The number of rows of the right matrix.
- **left_dtype** – Data type of the left matrix.

- **right_zp_dtype** – Data type of the right matrix and zero-point.

Remarks

- The output, left, right and zp start at the NPU 0, and all in 64-byte aligned layout.
- The shape of output is $[1, \text{left_rows}, 1, \text{right_rows}]$. The shape of left is $[1, \text{left_rows}, 1, \text{left_cols}]$. The shape of right is $[1, \text{right_rows}, 1, \text{left_cols}]$. The shape of zero-point is $[1, \text{NPU_NUM}, 1, \text{right_rows}]$, not $[1, 1, 1, \text{right_rows}]$. Each NPU has the same one.
- **left_rows**, **left_cols** and **right_rows** are in $[1, 65535]$.
- Data type of the output matrix is **DT_INT32**. The valid choices of **left_dtype** and **right_zp_dtype** are **DT_INT8** and **DT_UINT8**.

4.15.20 tpu_bdc_int8_pc_zp_mm_all_trans

Perform multiplication of the transposed left matrix and the transposed right matrix, and then perform result transposition and accumulation (optional) without result saturation. Subtract zero-point from the right matrix by row before transposition.

```
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)
```

$$\text{output}_{N \times M}^T = \text{output}_{N \times M}^T + \text{left}_{K \times M}^T \times (\text{right}_{N \times K} - \text{zp}_{N \times 1})^T$$

$$\text{output}(n, m) = \text{output}(n, m) + \sum_k \text{left}(k, m) \times (\text{right}(n, k) - \text{zp}(n, 0))$$

Parameters

- **output_addr** – Address of the output matrix.
- **left_addr** – Address of the left matrix.
- **right_addr** – Address of the right matrix.
- **zp_addr** – Address of the zero-point.
- **left_rows** – The number of rows of the left matrix.
- **left_cols** – The number of columns of the left matrix.
- **right_rows** – The number of rows of the right matrix.
- **left_dtype** – Data type of the left matrix.
- **right_zp_dtype** – Data type of the right matrix and **zp_val**.
- **result_add** – Flag of result accumulation.

Remarks

- The output, left, right and zp start at the NPU 0. The output, left and right are in 64-byte aligned layout. The zp is in compact layout.
- The shape of output is $[1, \text{right_rows}, 1, \text{left_cols}]$. The shape of left is $[1, \text{left_rows}, 1, \text{left_cols}]$. The shape of right is $[1, \text{right_rows}, 1, \text{left_rows}]$. The shape of zero-point is $[1, \text{right_rows}, 1, 1]$.
- **left_rows**, **left_cols** and **right_rows** are in $[1, 65535]$.

- Data type of the output matrix is *DT_INT32*. The valid choices of *left_dtype* and *right_zp_dtype* are *DT_INT8* and *DT_UINT8*.

4.15.21 tpu_bdc_int8_pc_zp_mm_L_const

Perform matrix multiplication and perform result accumulation (optional) without result saturation. Subtract zero-point from the right matrix by column.

```
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)
```

$$\text{output}_{M \times N} = \text{output}_{M \times N} + \text{left}_{M \times K} \times (\text{right}_{K \times N} - \text{zp}_{1 \times N})$$

$$\text{output}(m, n) = \text{output}(m, n) + \sum_k C \times (\text{right}(k, n) - \text{zp}(0, n))$$

Parameters

- **output_addr** – Address of the output matrix.
- **right_addr** – Address of the right matrix.
- **zp_addr** – Address of the zero-point.
- **C** – The constant value of the left matrix.
- **left_rows** – The number of rows of the left matrix.
- **left_cols** – The number of columns of the left matrix.
- **right_cols** – The number of columns of the right matrix.
- **C_dtype** – Data type of *C*.
- **right_zp_dtype** – Data type of the right matrix and *zp_val*.
- **result_add** – Flag of result accumulation.

Remarks

- The output, right and zp start at the NPU 0, and all in 64-byte aligned layout.
- The shape of output is [1, *left_rows*, 1, *right_cols*]. The shape of left is [1, *left_rows*, 1, *left_cols*]. The shape of right is [1, *left_cols*, 1, *right_cols*]. The shape of zero-point is [1, *NPU_NUM*, 1, *right_cols*], not [1, 1, 1, *right_cols*]. Each NPU has the same one.
- *left_rows*, *left_cols* and *right_cols* are in [1, 65535].
- Data type of the output matrix is *DT_INT32*. The valid choices of *C_dtype* and *right_zp_dtype* are *DT_INT8* and *DT_UINT8*.

4.15.22 tpu_bdc_int8_pc_zp_mm_R_const

Perform matrix multiplication and perform result accumulation (optional) without result saturation. Subtract zero-point from the right matrix by column. All elements of the right matrix have the same value.

```
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)
```

$$\text{output}_{M \times N} = \text{output}_{M \times N} + \text{left}_{M \times K} \times (\text{right}_{K \times N} - \text{zp}_{1 \times N})$$

$$\text{output}(m, n) = \text{output}(m, n) + \sum_k \text{left}(m, k) \times (C - \text{zp}(0, n))$$

Parameters

- **output_addr** – Address of the output matrix.
- **left_addr** – Address of the left matrix.
- **zp_addr** – Address of the zero-point.
- **C** – The constant value of the right matrix.
- **left_rows** – The number of rows of the left matrix.
- **left_cols** – The number of columns of the left matrix.
- **right_cols** – The number of columns of the right matrix.
- **left_dtype** – Data type of the left matrix.
- **C_zp_dtype** – Data type of the zero-point and *C*.
- **result_add** – Flag of result accumulation.

Remarks

- The output, left and zp start at the NPU 0, and all in 64-byte aligned layout.
- The shape of output is [1, left_rows, 1, right_cols]. The shape of left is [1, left_rows, 1, left_cols]. The shape of right is [1, left_cols, 1, right_cols]. The shape of zero-point is [1, *NPU_NUM*, 1, right_cols], not [1, 1, 1, right_cols]. Each NPU has the same one.
- left_rows, left_cols and right_cols are in [1, 65535].
- Data type of the output matrix is *DT_INT32*. The valid choices of left_dtype and C_zp_dtype are *DT_INT8* and *DT_UINT8*.

4.15.23 tpu_bdc_int8_pc_zp_mm_L_const_R_trans

Perform multiplication of the left matrix and the transposed right matrix without result saturation. Subtract zero-point from the transposed right matrix by column. All elements of the left matrix have the same value.

```
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)
```

$$\text{output}_{M \times N} = \text{left}_{M \times K} \times (\text{right}_{N \times K}^T - \text{zp}_{1 \times N})$$

$$\text{output}(m, n) = \sum_k C \times (\text{right}(n, k) - \text{zp}(0, n))$$

Parameters

- **output_addr** – Address of the output matrix.
- **right_addr** – Address of the right matrix.
- **zp_addr** – Address of the zero-point.
- **C** – The constant value of the left matrix.
- **left_rows** – The number of rows of the left matrix.
- **left_cols** – The number of columns of the left matrix.
- **right_rows** – The number of rows of the right matrix.
- **C_dtype** – Data type of *C*.
- **right_zp_dtype** – Data type of the right matrix and zero-point.

Remarks

- The output and right start at the NPU 0, and all in 64-byte aligned layout.
- The shape of output is [1, left_rows, 1, right_rows]. The shape of left is [1, left_rows, 1, left_cols]. The shape of right is [1, right_rows, 1, left_cols]. The shape of zero-point is [1, *NPU_NUM*, 1, right_rows], not [1, 1, 1, right_rows]. Each NPU has the same one.
- left_rows, left_cols and right_rows are in [1, 65535].
- Data type of the output matrix is *DT_INT32*. The valid choices of *C_dtype* and *right_zp_dtype* are *DT_INT8* and *DT_UINT8*.

4.15.24 tpu_bdc_int8_pc_zp_mm_L_const_all_trans

Perform multiplication of the transposed left matrix and the transposed right matrix, and then perform result transposition and accumulation (optional) without result saturation. Subtract zero-point from the right matrix by row before transposition. All elements of the left matrix have the same value.

```
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)
```

$$\text{output}_{N \times M}^T = \text{output}_{N \times M}^T + \text{left}_{K \times M}^T \times (\text{right}_{N \times K} - \text{zp}_{N \times 1})^T$$

$$\text{output}(n, m) = \text{output}(n, m) + \sum_k C \times (\text{right}(n, k) - \text{zp}(n, 0))$$

Parameters

- **output_addr** – Address of the output matrix.
- **right_addr** – Address of the right matrix.
- **zp_addr** – Address of the zero-point.
- **C** – The constant value of the left matrix.
- **left_rows** – The number of rows of the left matrix.
- **left_cols** – The number of columns of the left matrix.

- **right_rows** – The number of rows of the right matrix.
- **C_dtype** – Data type of *C*.
- **right_zp_dtype** – Data type of the right matrix and zero-point.
- **result_add** – Flag of result accumulation.

Remarks

- The output, right and zp start at the NPU 0. The output and right are in 64-byte aligned layout. The zp is in compact layout.
- The shape of output is [1, **right_rows**, 1, **left_cols**]. The shape of left is [1, **left_rows**, 1, **left_cols**]. The shape of right is [1, **right_rows**, 1, **left_rows**]. The shape of zero-point is [1, **right_rows**, 1, 1].
- **left_rows**, **left_cols** and **right_rows** are in [1, 65535].
- Data type of the output matrix is *DT_INT32*. The valid choices of **C_dtype** and **right_zp_dtype** are *DT_INT8* and *DT_UINT8*.

4.15.25 tpu_bdc_int8_pc_zp_mm_R_const_all_trans

Perform multiplication of the transposed left matrix and the transposed right matrix, and then perform result transposition and accumulation (optional) without result saturation. Subtract zero-point from the right matrix by row before transposition. All elements of the right matrix have the same value.

```
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)
```

$$\text{output}_{N \times M}^T = \text{output}_{N \times M}^T + \text{left}_{K \times M}^T \times (\text{right}_{N \times K} - \text{zp}_{N \times 1})^T$$

$$\text{output}(n, m) = \text{output}(n, m) + \sum_k \text{left}(k, m) \times (C - \text{zp}(n, 0))$$

Parameters

- **output_addr** – Address of the output matrix.
- **left_addr** – Address of the left matrix.
- **C** – The constant value of the right matrix.
- **zp_val** – Value of zero-point.
- **left_rows** – The number of rows of the left matrix.
- **left_cols** – The number of columns of the left matrix.
- **right_rows** – The number of rows of the right matrix.
- **left_dtype** – Data type of the left matrix.
- **C_zp_dtype** – Data type of the zero-point and *C*.
- **result_add** – Flag of result accumulation.

Remarks

- The output, left and zp start at the NPU 0. The output and left are in 64-byte aligned layout. The zp is in compact layout.

- The shape of output is `[1, right_rows, 1, left_cols]`. The shape of left is `[1, left_rows, 1, left_cols]`. The shape of right is `[1, right_rows, 1, left_rows]`. The shape of zero-point is `[1, right_rows, 1, 1]`.
- `left_rows`, `left_cols` and `right_rows` are in `[1, 65535]`.
- Data type of the output matrix is `DT_INT32`. The valid choices of `left_dtype` and `C_zp_dtype` are `DT_INT8` and `DT_UINT8`.

4.16 Floating Point Neural Network Functions

4.16.1 tpu_bdc_fp_bias

Perform adding bias to the source tensor on each channel.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **bias_addr** – Address of the bias tensor.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dtype** – Data type of the destination, source and bias tensors.

Remarks

- The destination, source and bias tensors start at the same NPU.
- The destination and source tensors are both in 64-byte aligned layout. The bias has the shape of `[1, shape->c, 1, 1]`, and is in compact layout.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are all in `[1, 65535]`.
- The valid choices of `dtype` are `DT_FP32`, `DT_FP16` and `DT_BFP16`.

4.16.2 tpu_bdc_fp_scale

Perform multiplication of the source tensor by the scale on each channel.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **scale_addr** – Address of the scale.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dtype** – Data type of the destination, source and scale tensors.

Remarks

- The destination, source and scale tensors start at the same NPU.
- The destination and source tensors are both in 64-byte aligned layout. The scale has the shape of $[1, \text{shape} \rightarrow c, 1, 1]$, and is in compact layout.
- $\text{shape} \rightarrow n$, $\text{shape} \rightarrow c$, $\text{shape} \rightarrow h$ and $\text{shape} \rightarrow w$ are all in $[1, 65535]$.
- The valid choices of `dtype` are *DT_FP32*, *DT_FP16* and *DT_BFP16*.

4.16.3 tpu_bdc_fp_scale_bias

Perform multiplication of the source tensor by the scale and adding bias on each channel.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **scale_addr** – Address of the scale.
- **bias_addr** – Address of the bias tensor.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dtype** – Data type of the destination, source, scale and bias tensors.

Remarks

- The destination, source, scale and bias tensors start at the same NPU.
- The destination and source tensors are both in 64-byte aligned layout. The bias and scale both have the shape of $[1, \text{shape} \rightarrow c, 1, 1]$, and are in compact layout.
- $\text{shape} \rightarrow n$, $\text{shape} \rightarrow c$, $\text{shape} \rightarrow h$ and $\text{shape} \rightarrow w$ are all in $[1, 65535]$.
- The valid choices of `dtype` are *DT_FP32*, *DT_FP16* and *DT_BFP16*.

4.16.4 tpu_bdc_fp_scale_bias_C

Perform multiplication of the source tensor by the scale and adding bias. The scale and bias both are constant.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **scale** – scale (constant value)

- **bias** – bias (constant value)
- **shape** – Pointer to the shape of the destination and source tensors.
- **dtype** – Data type of the destination, source, *scale* and *bias*.

Remarks

- The destination and source tensors start at the same NPU, and are both in 64-byte aligned layout.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are all in [1, 65535].
- The valid choices of **dtype** are *DT_FP32*, *DT_FP16* and *DT_BFP16*.

4.16.5 tpu_bdc_fp_add_bias_sqr

Perform adding bias to the source tensor on each channel, and square the result.

```
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)
```

$$\text{dst}(n, c, h, w) = (\text{src}(n, c, h, w) + \text{bias}(0, c, 0, 0))^2$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **bias_addr** – Address of the bias tensor.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dtype** – Data type of the destination, source and bias tensors.

Remarks

- The destination, source and bias tensors start at the same NPU.
- The destination and source tensors are both in 64-byte aligned layout. The bias has the shape of [1, **shape->c**, 1, 1], and is in compact layout.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are all in [1, 65535].
- The valid choices of **dtype** are *DT_FP32*, *DT_FP16* and *DT_BFP16*.

4.16.6 tpu_bdc_fp_add_C_sqr

Perform adding constant value to the source tensor on each channel, and square the result.

```
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)
```

$$\text{dst}(n, c, h, w) = (\text{src}(n, c, h, w) + C)^2$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **C** – Constant value.
- **shape** – Pointer to the shape of the destination and source tensors.

- **dtype** – Data type of the destination, source and C .

Remarks

- The destination and source tensors start at the same NPU, and are both in 64-byte aligned layout.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are all in $[1, 65535]$.
- The valid choices of `dtype` are `DT_FP32`, `DT_FP16` and `DT_BFP16`.

4.16.7 tpu_bdc_fp_sub_bias_sqr

Perform subtracting the source tensor on each channel by bias, and square the result.

```
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)
```

$$\text{dst}(n, c, h, w) = (\text{src}(n, c, h, w) - \text{bias}(0, c, 0, 0))^2$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **bias_addr** – Address of the bias tensor.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dtype** – Data type of the destination, source and bias tensors.

Remarks

- The destination, source and bias tensors start at the same NPU.
- The destination and source tensors are both in 64-byte aligned layout. The bias has the shape of $[1, \text{shape->c}, 1, 1]$, and is in compact layout.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are all in $[1, 65535]$.
- The valid choices of `dtype` are `DT_FP32`, `DT_FP16` and `DT_BFP16`.

4.16.8 tpu_bdc_fp_sub_C_sqr

Perform subtracting the source tensor on each channel by a constant, and square the result.

```
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)
```

$$\text{dst}(n, c, h, w) = (\text{src}(n, c, h, w) - C)^2$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **C** – Constant value.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dtype** – Data type of the destination, source and C .

Remarks

- The destination and source tensors start at the same NPU, and are both in 64-byte aligned layout.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are all in `[1, 65535]`.
- The valid choices of `dtype` are `DT_FP32`, `DT_FP16` and `DT_BFP16`.

4.16.9 tpu_bdc_fp_conv2d

Perform 2D convolution with adding bias (optional) and accumulation on result (optional).

```
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)
```

Parameters

- **output_addr** – Address of the output tensor.
- **input_addr** – Address of the input tensor.
- **weight_addr** – Address of the weight tensor.
- **bias_addr** – Address of the bias tensor.
- **input_shape** – Pointer to the shape of the input tensor.
- **input_stride** – Pointer to the stride of the input tensor.
- **output_c** – The number of output channels.
- **kernel** – Pointer to the size of the kernel.
- **padding** – Pointer to the size of the padding.
- **stride** – Pointer to the stride.
- **dilation** – Pointer to the dilation.
- **output_dtype** – Data type of output tensor.
- **input_dtype** – The data type of the input and weight. If `has_bias` is true, the data type of the bias will be recognized as `DT_FP32`.
- **has_bias** – Flag of `has_bias`.
- **result_add** – Flag of `result_add`.

Remarks

- The output, weight and bias tensors start at the same NPU, and input tensor starts at NPU 0.
- The shape of output is `[input_shape->n, output_c, output_h, output_w]`, 64-byte aligned layout,

$$\text{output_h} = (\text{input_shape}\text{->h} + \text{padding}\text{->top} + \text{padding}\text{->bottom} - ((\text{kernel}\text{->h} - 1) * \text{dilation}\text{->h} + 1)) / \text{stride}\text{->h} + 1,$$

$$\text{output_w} = (\text{input_shape}\text{->w} + \text{padding}\text{->left} + \text{padding}\text{->right} - ((\text{kernel}\text{->w} - 1) * \text{dilation}\text{->w} + 1)) / \text{stride}\text{->w} + 1.$$

- The input is in free layout. The bias has the shape of `[1, output_c, 1, 1]`, and is in compact layout. The shape of weight is `[input_shape->c, output_c, kernel->h, kernel->w]`. If `input_dtype` is `DT_FP32`, then the weight is in compact layout, otherwise in 32-IC layout.

- `input_shape->n`, `input_shape->c`, `input_shape->h`, `input_shape->w`, `kernel->h` and `kernel->w` are in [1, 65535].
- The valid choices of `input_dtype` are `DT_FP32`, `DT_FP16` and `DT_BFP16`. The `output_dtype` is same as `input_dtype` or `DT_FP32`.
- `padding->top`, `padding->bottom`, `padding->left` and `padding->right` are in [0, 15]. `stride->h`, `stride->w`, `dilation->h` and `dilation->w` are in [1, 15].

4.16.10 tpu_bdc_fp_conv2d_kernel_const

Perform constant-kernel 2D convolution with adding bias (optional) and accumulation on result (optional).

```
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)
```

Parameters

- **output_addr** – Address of the output tensor.
- **input_addr** – Address of the input tensor.
- **bias_addr** – Address of the bias tensor.
- **C** – Constant value of the weight tensor.
- **input_shape** – Pointer to the shape of the input tensor.
- **input_stride** – Pointer to the stride of the input tensor.
- **output_c** – The number of output channels.
- **kernel** – Pointer to the size of the kernel.
- **padding** – Pointer to the size of the padding.
- **stride** – Pointer to the stride.
- **dilation** – Pointer to the dilation.
- **output_dtype** – Data type of output tensor.
- **input_dtype** – The data type of the input and weight. If `has_bias` is true, the data type of the bias will be recognized as `DT_FP32`.
- **has_bias** – Flag of `has_bias`.
- **result_add** – Flag of `result_add`.

Remarks

- The output, weight and bias tensors start at the same NPU, and input tensor starts at NPU 0.
- The shape of output is [`input_shape->n`, `output_c`, `output_h`, `output_w`], 64-byte aligned layout,

$$\text{output_h} = (\text{input_shape->h} + \text{padding->top} + \text{padding->bottom} - ((\text{kernel->h} - 1) * \text{dilation->h} + 1)) / \text{stride->h} + 1,$$

```
output_w = (input_shape->w + padding->left + padding->right - ((kernel->w - 1) * dilation->w + 1)) / stride->w + 1.
```

- The input is in free layout. The bias has the shape of [1, output_c, 1, 1], and is in compact layout. The shape of weight is [input_shape->c, output_c, kernel->h, kernel->w].
- input_shape->n, input_shape->c, input_shape->h, input_shape->w, kernel->h and kernel->w are in [1, 65535].
- The valid choices of input_dtype are *DT_FP32*, *DT_FP16* and *DT_BFP16*. The output_dtype is same as input_dtype or *DT_FP32*.
- padding->top, padding->bottom, padding->left and padding->right are in [0, 15]. stride->h, stride->w, dilation->h and dilation->w are in [1, 15].

4.16.11 tpu_bdc_fp_conv2d_for_deconv2d

Perform 2D convolution to achieve 2D deconvolution. The input tensor supports insertion, and the kernel should be rotated. Perform the result with accumulation (optional) and adding bias on each channel (optional).

```
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)
```

Parameters

- **output_addr** – Address of the output tensor.
- **input_addr** – Address of the input tensor.
- **weight_addr** – Address of the weight tensor.
- **bias_addr** – Address of the bias tensor.
- **input_shape** – Pointer to the shape of the input tensor.
- **input_stride** – Pointer to the stride of the input tensor.
- **output_c** – The number of output channels.
- **kernel** – Pointer to the size of the kernel.
- **insert** – Pointer to the insert.
- **padding** – Pointer to the size of the padding.
- **dilation** – Pointer to the dilation.
- **output_dtype** – Data type of output tensor.
- **input_dtype** – The data type of the input and weight. If has_bias is true, the data type of the bias will be recognized as *DT_FP32*.
- **has_bias** – Flag of has_bias.
- **result_add** – Flag of result_add.

Remarks

- The output, weight and bias tensors start at the same NPU, and input tensor starts at NPU 0.

- The shape of output is `[input_shape->n, output_c, output_h, output_w]`, 64-byte aligned layout,

$$\text{output_h} = ((\text{input_shape}->\text{h} - 1) * (\text{insert}->\text{h} + 1) + 1 + \text{padding}->\text{top} + \text{padding}->\text{bottom} - ((\text{kernel}->\text{h} - 1) * \text{dilation}->\text{h} + 1)) / \text{stride}->\text{h} + 1,$$

$$\text{output_w} = ((\text{input_shape}->\text{w} - 1) * (\text{insert}->\text{w} + 1) + 1 + \text{padding}->\text{left} + \text{padding}->\text{right} - ((\text{kernel}->\text{w} - 1) * \text{dilation}->\text{w} + 1)) / \text{stride}->\text{w} + 1.$$
- The input is in free layout. The bias has the shape of `[1, output_c, 1, 1]`, and is in compact layout. The shape of weight is `[input_shape->c, output_c, kernel->h, kernel->w]`. If `input_dtype` is `DT_FP32`, then the weight is in compact layout, otherwise in 32-IC layout.
- `input_shape->n`, `input_shape->c`, `input_shape->h`, `input_shape->w`, `kernel->h` and `kernel->w` are in `[1, 65535]`.
- The valid choices of `input_dtype` are `DT_FP32`, `DT_FP16` and `DT_BFP16`. The `output_dtype` is same as `input_dtype` or `DT_FP32`.
- `padding->top`, `padding->bottom`, `padding->left` and `padding->right` are in `[0, 15]`. `insert->h` and `insert->w` are in `[0, 14]`. `dilation->h` and `dilation->w` are in `[1, 15]`.

4.16.12 tpu_bdc_fp_max_pool2d

Perform 2D max pooling, the value of padding can be customized.

```
void tpu_bdc_fp_max_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 dtype, scalar_t pad_val)
```

Parameters

- **output_addr** – Address of the output tensor.
- **input_addr** – Address of the input tensor.
- **input_shape** – Pointer to the shape of the input tensor.
- **kernel** – Pointer to the size of the kernel.
- **padding** – Pointer to the padding.
- **stride** – Pointer to the stride.
- **dilation** – Pointer to the dilation.
- **dtype** – The data type of the output, input and *pad_val*.
- **pad_val** – Value of padding.

Remarks

- The output and input tensors start at the same NPU, and are both in 64-byte aligned layout.
- The shape of output is `[input_shape->n, input_shape->c, output_h, output_w]`,

$$\text{output_h} = (\text{input_shape}->\text{h} + \text{padding}->\text{top} + \text{padding}->\text{bottom} - ((\text{kernel}->\text{h} - 1) * \text{dilation}->\text{h} + 1)) / \text{stride}->\text{h} + 1,$$

$$\text{output_w} = (\text{input_shape}->\text{w} + \text{padding}->\text{left} + \text{padding}->\text{right} - ((\text{kernel}->\text{w} - 1) * \text{dilation}->\text{w} + 1)) / \text{stride}->\text{w} + 1.$$
- `input_shape->n`, `input_shape->c`, `input_shape->h`, `input_shape->w`, `kernel->h` and `kernel->w` are in `[1, 65535]`.

- The valid choices of `dtype` are `DT_FP32`, `DT_FP16` and `DT_BFP16`.
- `padding->top`, `padding->bottom`, `padding->left` and `padding->right` are in $[0, 15]$.
`stride->h`, `stride->w`, `dilation->h` and `dilation->w` are in $[1, 15]$.

4.16.13 tpu_bdc_fp_ins_avg_pool2d

Perform 2D average pooling with insertion, the value of scale can be customized (replace `1 / (kernel->h * 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)
```

Parameters

- `output_addr` – Address of the output tensor.
- `input_addr` – Address of the input tensor.
- `input_shape` – Pointer to the shape of the input tensor.
- `kernel` – Pointer to the size of the kernel.
- `padding` – Pointer to the padding.
- `stride` – Pointer to the stride.
- `dilation` – Pointer to the dilation.
- `ins` – Pointer to the insertion (The number of elements inserted among the rows/columns of the input feature map).
- `dtype` – The data type of the output, input and `scale`.
- `scale` – Value of the scale.

Remarks

- The output and input tensors start at the same NPU, and are both in 64-byte aligned layout.
- The shape of output is $[\text{input_shape} \rightarrow n, \text{input_shape} \rightarrow c, \text{output_h}, \text{output_w}]$,

$$\text{output_h} = (\text{input_shape} \rightarrow h + \text{padding} \rightarrow \text{top} + \text{padding} \rightarrow \text{bottom} - ((\text{kernel} \rightarrow h - 1) * \text{dilation} \rightarrow h + 1)) / \text{stride} \rightarrow h + 1,$$

$$\text{output_w} = (\text{input_shape} \rightarrow w + \text{padding} \rightarrow \text{left} + \text{padding} \rightarrow \text{right} - ((\text{kernel} \rightarrow w - 1) * \text{dilation} \rightarrow w + 1)) / \text{stride} \rightarrow w + 1.$$
- `input_shape->n`, `input_shape->c`, `input_shape->h`, `input_shape->w`, `kernel->h` and `kernel->w` are in $[1, 65535]$.
- The valid choices of `dtype` are `DT_FP32`, `DT_FP16` and `DT_BFP16`.
- `padding->top`, `padding->bottom`, `padding->left` and `padding->right` are in $[0, 15]$.
`stride->h`, `stride->w`, `dilation->h` and `dilation->w` are in $[1, 15]$.
- `ins->h`, `ins->w` are in $[0, 8)$.

4.16.14 tpu_bdc_fp_avg_pool2d

Perform 2D average pooling, the value of scale can be customized (replace `1 / (kernel->h * 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)
```

Parameters

- **output_addr** – Address of the output tensor.
- **input_addr** – Address of the input tensor.
- **input_shape** – Pointer to the shape of the input tensor.
- **kernel** – Pointer to the size of the kernel.
- **padding** – Pointer to the padding.
- **stride** – Pointer to the stride.
- **dilation** – Pointer to the dilation.
- **dtype** – The data type of the output, input and *scale*.
- **scale** – Value of the scale.

Remarks

- The output and input tensors start at the same NPU, and are both in 64-byte aligned layout.
- The shape of output is `[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` and `kernel->w` are in `[1, 65535]`.
- The valid choices of `dtype` are *DT_FP32*, *DT_FP16* and *DT_BFP16*.
- `padding->top`, `padding->bottom`, `padding->left` and `padding->right` are in `[0, 15]`.
`stride->h`, `stride->w`, `dilation->h` and `dilation->w` are in `[1, 15]`.

4.16.15 tpu_bdc_fp_depthwise2d

Perform 2D depthwise convolution with adding bias (optional), and then perform result accumulation (optional).

```
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)
```

Parameters

- **output_addr** – Address of the output tensor.
- **input_addr** – Address of the input tensor.

- **weight_addr** – Address of the weight tensor.
- **bias_addr** – Address of the bias tensor.
- **input_shape** – Pointer to the shape of the input tensor.
- **kernel** – Pointer to the size of the kernel.
- **padding** – Pointer to the padding.
- **stride** – Pointer to the stride.
- **dilation** – Pointer to the dilation.
- **dtype** – The data type of the output, input weight and bias.
- **has_bias** – Flag of has_bias.

Remarks

- The output, input, weight and bias tensors start at the same NPU.
- The shape of output is [input_shape->n, input_shape->c, output_h, output_w], 64-byte aligned layout,

$$\text{output_h} = (\text{input_shape}\text{->h} + \text{padding}\text{->top} + \text{padding}\text{->bottom} - ((\text{kernel}\text{->h} - 1) * \text{dilation}\text{->h} + 1)) / \text{stride}\text{->h} + 1,$$

$$\text{output_w} = (\text{input_shape}\text{->w} + \text{padding}\text{->left} + \text{padding}\text{->right} - ((\text{kernel}\text{->w} - 1) * \text{dilation}\text{->w} + 1)) / \text{stride}\text{->w} + 1.$$
- The input is in 64-byte aligned layout. The weight has the shape of [1, input_shape->c, kernel->h, kernel->w], and is in compact layout. The bias has the shape of [1, input_shape->c, 1, 1], and is in compact layout.
- input_shape->n, input_shape->c, input_shape->h, input_shape->w, kernel->h and kernel->w are in [1, 65535].
- The valid choices of dtype are *DT_FP32*, *DT_FP16* and *DT_BFP16*.
- padding->top, padding->bottom, padding->left and padding->right are in [0, 15]. stride->h, stride->w, dilation->h and dilation->w are in [1, 15].

4.17 Fixed Point Neural Network Functions

4.17.1 tpu_bdc_int8_max_pool2d

Perform 2D max pooling, the value of padding can be customized.

```
void tpu_bdc_int8_max_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 dtype, scalar_t pad_val)
```

Parameters

- **output_addr** – Address of the output tensor.
- **input_addr** – Address of the input tensor.
- **input_shape** – Pointer to the shape of the input tensor.
- **kernel** – Pointer to the size of the kernel.
- **padding** – Pointer to the size of the padding.
- **stride** – Pointer to the stride.

- **dilation** – Pointer to the dilation.
- **dtype** – The data type of the output, input and *pad_val*.
- **pad_val** – Value of padding.

Remarks

- The output and input tensors start at the same NPU, and are both in 64-byte aligned layout.
- The shape of output is $[\text{input_shape} \rightarrow n, \text{input_shape} \rightarrow c, \text{output_h}, \text{output_w}]$,
 $\text{output_h} = (\text{input_shape} \rightarrow h + \text{padding} \rightarrow \text{top} + \text{padding} \rightarrow \text{bottom} - ((\text{kernel} \rightarrow h - 1) * \text{dilation} \rightarrow h + 1)) / \text{stride} \rightarrow h + 1$,
 $\text{output_w} = (\text{input_shape} \rightarrow w + \text{padding} \rightarrow \text{left} + \text{padding} \rightarrow \text{right} - ((\text{kernel} \rightarrow w - 1) * \text{dilation} \rightarrow w + 1)) / \text{stride} \rightarrow w + 1$.
- $\text{input_shape} \rightarrow n, \text{input_shape} \rightarrow c, \text{input_shape} \rightarrow h, \text{input_shape} \rightarrow w, \text{kernel} \rightarrow h$ and $\text{kernel} \rightarrow w$ are in $[1, 65535]$.
- The valid choices of **dtype** are *DT_INT8* and *DT_UINT8*.
- $\text{padding} \rightarrow \text{top}, \text{padding} \rightarrow \text{bottom}, \text{padding} \rightarrow \text{left}$ and $\text{padding} \rightarrow \text{right}$ are in $[0, 15]$.
 $\text{stride} \rightarrow h, \text{stride} \rightarrow w, \text{dilation} \rightarrow h$ and $\text{dilation} \rightarrow w$ are in $[1, 15]$.

4.17.2 tpu_bdc_int8_avg_pool2d

Perform 2D average pooling, the value of padding can be customized. Then perform arithmetic shift and saturation (optional) on the result.

```
void tpu_bdc_int8_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 output_dtype, data_type_t
    input_dtype, unsigned char scale, unsigned char rshift)
```

Parameters

- **output_addr** – Address of the output tensor.
- **input_addr** – Address of the input tensor.
- **input_shape** – Pointer to the shape of the input tensor.
- **kernel** – Pointer to the size of the kernel.
- **padding** – Pointer to the padding.
- **stride** – Pointer to the stride.
- **dilation** – Pointer to the dilation.
- **output_dtype** – The data type of the output.
- **input_dtype** – The data type of the input and *scale*.
- **scale** – Value of the scale.
- **rshift** – Number of right shifts.

Remarks

- The output and input tensors start at the same NPU, and are both in 64-byte aligned layout.

- The shape of output is `[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` and `kernel->w` are in `[1, 65535]`.
- The valid choices of `output_dtype` are `DT_INT32`, `DT_UINT32`, `DT_INT16`, `DT_UINT16`, `DT_INT8` and `DT_UINT8`. The valid choices of `input_dtype` are `DT_INT8` and `DT_UINT8`. The `output_dtype` and `input_dtype` have the same sign.
- `padding->top`, `padding->bottom`, `padding->left` and `padding->right` are in `[0, 15]`, `stride->h`, `stride->w`, `dilation->h` and `dilation->w` are in `[1, 15]`.
- `rshift` are in `[0, 31]`.

4.18 Activation Functions

4.18.1 tpu_bdc_relu

Calculate *ReLU* of the source tensor.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{src}(n, c, h, w) & \text{if } \text{shift}(n, c, h, w) > 0 \\ 0 & \text{Otherwise.} \end{cases}$$

Parameters

- `dst_addr` – Address of the destination tensor.
- `src_addr` – Address of the source tensor.
- `shape` – Pointer to the shape of the destination and source tensors.
- `dst_stride` – Pointer to the stride of the destination tensor.
- `src_stride` – Pointer to the stride of the source tensor.
- `dtype` – Data type of source and destination tensors.

Remarks

- The destination and source tensors start at the same NPU.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in `[1, 65535]`.
- If `dst_stride` or `src_stride` is `NULL`, the relative tensor is in 64-byte aligned layout, otherwise in free layout.

4.18.2 tpu_bdc_prelu

Calculate *PReLU* of the source tensor.

```
void tpu_bdc_prelu(local_addr_t dst_addr, local_addr_t src_addr, scalar_t alpha, const
                  dim4 *shape, data_type_t dtype)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{src}(n, c, h, w) & \text{if } \text{shift}(n, c, h, w) > 0 \\ \alpha \times \text{src}(n, c, h, w) & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **alpha** – The constant value.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dtype** – Data type of source and destination tensors and *alpha*.

Remarks

- The destination and source tensors start at the same NPU, and both are in 64-byte aligned layout.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535].

4.18.3 tpu_bdc_fp32_elu

Calculate *ELU* of the source tensor.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} \text{src}(n, c, h, w) & \text{if } \text{shift}(n, c, h, w) > 0 \\ \alpha \times (e^{\text{src}(n, c, h, w)} - 1) & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **work0_addr** – Address of the work0.
- **work1_addr** – Address of the work1.
- **coeff_addr** – Address of the coeff.
- **table_addr** – Address of the table.
- **alpha** – The constant value.
- **shape** – Pointer to the shape of the dst, src, work0 and work1.

Remarks

- The dst, src, work0 and work1 start at the same NPU, and all are in 64-byte aligned layout.
- The coeff is loaded by *tpu_bdc_load_fp32_exp_coeff()*, and the table is loaded by *tpu_bdc_load_fp32_exp_table()*.

- The work0 and work1 tensors are workspaces to store temporary tensors. Any two of `dst_addr`, `src_addr`, `work0_addr` and `work1_addr` are not allowed to be equal. If there is no bank conflicting between any two of destination, work1 and table tensors, the performance will be better.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in $[1, 65535]$. `shape->h * shape->w` less than or equal to 65535.
- If `src(n, c, h, w)` is less than -103 or greater than 88, then $e^{\text{src}(n,c,h,w)}$ will be e^{-103} or e^{88} , respectively.

4.18.4 tpu_bdc_fp32_sigmoid

Calculate *sigmoid* of the source tensor.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **work0_addr** – Address of the work0.
- **work1_addr** – Address of the work1.
- **coeff_addr** – Address of the coeff.
- **table_addr** – Address of the table.
- **shape** – Pointer to the shape of the dst, src, work0 and work1.

Remarks

- The dst, src, work0 and work1 start at the same NPU, and all are in 64-byte aligned layout.
- The coeff is loaded by `tpu_bdc_load_fp32_exp_coeff()`, and the table is loaded by `tpu_bdc_load_fp32_exp_table()`.
- The work0 and work1 tensors are workspaces to store temporary tensors. Any two of `dst_addr`, `work0_addr` and `work1_addr` are not allowed to be equal. Allow `src_addr` to be equal to `dst_addr`, `work0_addr` or `work1_addr`, in this case, source data will be overwritten. If there is no bank conflicting between any two of destination, work1 and table tensors, the performance will be better.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in $[1, 65535]$. `shape->h * shape->w` less than or equal to 65535.
- If `src(n, c, h, w)` is less than -88 or greater than 103, then $e^{-\text{src}(n,c,h,w)}$ will be e^{88} or e^{-103} , respectively.

4.18.5 tpu_bdc_fp32_tanh

Calculate *tanh* of the source tensor.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **work0_addr** – Address of the work0.
- **work1_addr** – Address of the work1.
- **coeff_addr** – Address of the coeff.
- **table_addr** – Address of the table.
- **shape** – Pointer to the shape of the dst, src, work0 and work1.

Remarks

- The dst, src, work0 and work1 start at the same NPU, and all are in 64-byte aligned layout.
- The coeff is loaded by *tpu_bdc_load_fp32_exp_coeff()*, and the table is loaded by *tpu_bdc_load_fp32_exp_table()*.
- The work0 and work1 tensors are workspaces to store temporary tensors. Any two of dst_addr, work0_addr and work1_addr are not allowed to be equal. Allow src_addr to be equal to dst_addr, work0_addr or work1_addr, in this case, source data will be overwritten. If there is no bank conflicting between any two of destination, work1 and table tensors, the performance will be better.
- shape->n, shape->c, shape->h and shape->w are in [1, 65535]. shape->h * shape->w less than or equal to 65535.
- If src(n, c, h, w) is less than -103 or greater than 88, then $e^{\text{src}(n, c, h, w)}$ will be e^{-103} or e^{88} respectively. If src(n, c, h, w) is less than -88 or greater than 103, then $e^{-\text{src}(n, c, h, w)}$ will be e^{88} or e^{-103} , respectively.

4.18.6 tpu_bdc_fp32_softplus

Calculate *softplus* of the source tensor.

```
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)
```

$$\text{dst}(n, c, h, w) = \frac{1}{\beta} \log(1 + e^{\beta \times \text{src}(n, c, h, w)})$$

Parameters

- **dst_addr** – Address of the destination tensor.

- **src_addr** – Address of the source tensor.
- **work0_addr** – Address of the work0.
- **work1_addr** – Address of the work1.
- **exp_coeff_addr** – exp_ Address of the coeff.
- **log_coeff_addr** – log_ Address of the coeff.
- **exp_table_addr** – exp_ Address of the table.
- **shape** – Pointer to the shape of the dst, src, work0 and work1.

Remarks

- The dst, src, work0 and work1 start at the same NPU, and all are in 64-byte aligned layout.
- The exp_coeff is loaded by `tpu_bdc_load_fp32_exp_coeff()` . The log_coeff is loaded by `tpu_bdc_load_fp32_log_coeff()` . The exp_table is loaded by `tpu_bdc_load_fp32_exp_table()` .
- The work0 and work1 tensors are workspaces to store temporary tensors. Any two of **dst_addr**, **work0_addr** and **work1_addr** are not allowed to be equal. Allow **src_addr** to be equal to **dst_addr**, **work0_addr** or **work1_addr**, in this case, source data will be overwritten. If there is no bank conflicting between any two of destination, work1 and table tensors, the performance will be better.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535]. **shape->h * shape->w** less than or equal to 65535.
- If $\beta \times \text{src}(n, c, h, w)$ is less than -103 or greater than 88, then $e^{\beta \times \text{src}(n, c, h, w)}$ will be e^{-103} or e^{88} , respectively.

4.18.7 tpu_bdc_fp32_erf

Calculate *erf* of the source tensor.

```
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)
```

$$\text{dst}(n, c, h, w) = \frac{2}{\sqrt{\pi}} \int_0^{\text{src}(n, c, h, w)} e^{-t^2} dt$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **work0_addr** – Address of the work0.
- **work1_addr** – Address of the work1.
- **work2_addr** – Address of the work2.
- **exp_coeff_addr** – exp_ Address of the coeff.
- **erf_coeff_addr** – erf_ Address of the coeff.
- **exp_table_addr** – exp_ Address of the table.
- **shape** – Pointer to the shape of the dst, src, work0, work1 and work2.

Remarks

- The dst, src, work0 ,work1 and work2 start at the same NPU, and all are in 64-byte aligned layout.
- The exp_coeff is loaded by `tpu_bdc_load_fp32_exp_coeff()` . The erf_coeff is loaded by `tpu_bdc_load_fp32_erf_coeff()` . The exp_table is loaded by `tpu_bdc_load_fp32_exp_table()` .
- The work0, work1 and work2 tensors are workspaces to store temporary tensors. Any two of dst_addr, src_addr, work0_addr , work1_addr and work2_addr are not allowed to be equal. If there is no bank conflicting between any two of destination, work1 and exp_table, the performance will be better.
- shape->n, shape->c, shape->h and shape->w are in [1, 65535]. shape->h * shape->w less than or equal to 65535.
- If e^t is less than -103 or greater than 88, then e^t will be e^{-103} or e^{88} , respectively.

4.18.8 tpu_bdc_fp32_erfc

Calculate *erfc* of the source tensor.

```
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)
```

$$\text{dst}(n, c, h, w) = 1 - \frac{2}{\sqrt{\pi}} \int_0^{\text{src}(n, c, h, w)} e^{-t^2} dt$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **work0_addr** – Address of the work0.
- **work1_addr** – Address of the work1.
- **work2_addr** – Address of the work2.
- **exp_coeff_addr** – exp_Address of the coeff.
- **erf_coeff_addr** – erf_Address of the coeff.
- **exp_table_addr** – exp_Address of the table.
- **shape** – Pointer to the shape of the dst, src, work0, work1 and work2.

Remarks

- The dst, src, work0 ,work1 and work2 start at the same NPU, and all are in 64-byte aligned layout.
- The exp_coeff is loaded by `tpu_bdc_load_fp32_exp_coeff()` . The erf_coeff is loaded by `tpu_bdc_load_fp32_erf_coeff()` . The exp_table is loaded by `tpu_bdc_load_fp32_exp_table()` .
- The work0, work1 and work2 tensors are workspaces to store temporary tensors. Any two of dst_addr, src_addr, work0_addr , work1_addr and work2_addr are not allowed to be equal. If there is no bank conflicting between any two of destination, work1 and exp_table, the performance will be better.

- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in $[1, 65535]$. `shape->h * shape->w` less than or equal to 65535.
- If e^t is less than -103 or greater than 88, then e^t will be e^{-103} or e^{88} , respectively.

4.18.9 tpu_bdc_fp32_gelu

Calculate *GELU* of the source tensor.

```
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)
```

$$\text{dst}(n, c, h, w) = \frac{1}{2} \left(1 + \text{erf} \left(\frac{\text{src}(n, c, h, w)}{\sqrt{2}} \right) \right)$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **work0_addr** – Address of the work0.
- **work1_addr** – Address of the work1.
- **work2_addr** – Address of the work2.
- **work3_addr** – Address of the work3.
- **exp_coeff_addr** – exp_ Address of the coeff.
- **erf_coeff_addr** – erf_ Address of the coeff.
- **exp_table_addr** – exp_ Address of the table.
- **shape** – Pointer to the shape of the dst, src, work0, work1, work2 and work3.

Remarks

- The dst, src, work0, work1, work2 and work3 start at the same NPU, and all are in 64-byte aligned layout.
- The exp_coeff is loaded by `tpu_bdc_load_fp32_exp_coeff()`. The erf_coeff is loaded by `tpu_bdc_load_fp32_erf_coeff()`. The exp_table is loaded by `tpu_bdc_load_fp32_exp_table()`.
- The work0, work1, work2 and work3 tensors are workspaces to store temporary tensors. Any two of `dst_addr`, `work0_addr`, `work1_addr`, `work2_addr` and `work3_addr` are not allowed to be equal. Allow `src_addr` to be equal to `dst_addr`, `work0_addr`, `work1_addr`, `work2_addr` or `work3_addr`, in this case, source data will be overwritten. If there is no bank conflicting between any two of destination, work1 and exp_table, the performance will be better.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in $[1, 65535]$. `shape->h * shape->w` less than or equal to 65535.
- The *erf* refers to `tpu_bdc_fp32_erf()`.

4.18.10 tpu_bdc_fp32_gelu_fast

Calculate *fastGELU* of the source tensor.

```
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)
```

$$\text{dst}(n, c, h, w) = \frac{\text{src}(n, c, h, w)}{2} \times \left(1 + \tanh \left(\sqrt{\frac{2}{\pi}} \times (\text{src}(n, c, h, w) + 0.044715 \times \text{src}(n, c, h, w)^3) \right) \right)$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **work0_addr** – Address of the work0.
- **work1_addr** – Address of the work1.
- **coeff_addr** – Address of the coeff.
- **table_addr** – Address of the table.
- **shape** – Pointer to the shape of the dst, src, work0 and work1.

Remarks

- The dst, src, work0 and work1 start at the same NPU, and all are in 64-byte aligned layout.
- The coeff is loaded by *tpu_bdc_load_fp32_exp_coeff()*, and the table is loaded by *tpu_bdc_load_fp32_exp_table()*.
- The work0 and work1 tensors are workspaces to store temporary tensors. Any two of **dst_addr**, **src_addr**, **work0_addr** and **work1_addr** are not allowed to be equal. If there is no bank conflicting between any two of destination, work1 and table tensors, the performance will be better.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535]. **shape->h * shape->w** less than or equal to 65535.
- The *tanh* refers to *tpu_bdc_fp32_tanh()*.

4.18.11 tpu_bdc_fp32_mish

Calculate *Mish* of the source tensor.

```
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)
```

$$\begin{aligned} \text{dst}(n, c, h, w) &= \text{src}(n, c, h, w) \times \tanh(\log(1 + e^{\text{src}(n, c, h, w)})) \\ &= \text{src}(n, c, h, w) \times \left(1 - 2 \times \frac{1}{(1 + e^{\text{src}(n, c, h, w)})^2 + 1} \right) \end{aligned}$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.

- **work0_addr** – Address of the work0.
- **work1_addr** – Address of the work1.
- **coeff_addr** – Address of the coeff.
- **table_addr** – Address of the table.
- **shape** – Pointer to the shape of the dst, src, work0 and work1.

Remarks

- The dst, src, work0 and work1 start at the same NPU, and all are in 64-byte aligned layout.
- The coeff is loaded by `tpu_bdc_load_fp32_exp_coeff()`, and the table is loaded by `tpu_bdc_load_fp32_exp_table()`.
- The work0 and work1 tensors are workspaces to store temporary tensors. Any two of **dst_addr**, **src_addr**, **work0_addr** and **work1_addr** are not allowed to be equal. If there is no bank conflicting between any two of destination, work1 and table tensors, the performance will be better.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are in [1, 65535]. **shape->h * shape->w** less than or equal to 65535.
- If **src(n, c, h, w)** is less than -103 or greater than 88, then $e^{\text{src}(n, c, h, w)}$ will be e^{-103} or e^{88} respectively.

4.18.12 tpu_bdc_fp32_silu

Calculate *SiLU* of the source tensor.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **work0_addr** – Address of the work0.
- **work1_addr** – Address of the work1.
- **coeff_addr** – Address of the coeff.
- **table_addr** – Address of the table.
- **shape** – Pointer to the shape of the dst, src, work0 and work1.

Remarks

- The dst, src, work0 and work1 start at the same NPU, and all are in 64-byte aligned layout.
- The coeff is loaded by `tpu_bdc_load_fp32_exp_coeff()`, and the table is loaded by `tpu_bdc_load_fp32_exp_table()`.
- The work0 and work1 tensors are workspaces to store temporary tensors. Any two of **dst_addr**, **src_addr**, **work0_addr** and **work1_addr** are not allowed to be equal. If there is no bank conflicting between any two of destination, work1 and table tensors, the performance will be better.

- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in $[1, 65535]$. `shape->h * shape->w` less than or equal to 65535.
- If `src(n, c, h, w)` is less than -88 or greater than 103, then $e^{-\text{src}(n,c,h,w)}$ will be e^{88} or e^{-103} , respectively.

4.18.13 tpu_bdc_fp32_selu

Calculate *SELU* of the source tensor.

```
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)
```

$$\text{dst}(n, c, h, w) = \lambda \times \begin{cases} \text{src}(n, c, h, w) & \text{if } \text{shift}(n, c, h, w) > 0 \\ \alpha \times (e^{\text{src}(n,c,h,w)} - 1) & \text{Otherwise.} \end{cases}$$

$$\alpha = 1.6732632423543772848170429916717$$

$$\lambda = 1.0507009873554804934193349852946$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **work0_addr** – Address of the work0.
- **work1_addr** – Address of the work1.
- **coeff_addr** – Address of the coeff.
- **table_addr** – Address of the table.
- **alpha** – The constant value.
- **shape** – Pointer to the shape of the dst, src, work0 and work1.

Remarks

- The dst, src, work0 and work1 start at the same NPU, and all are in 64-byte aligned layout.
- The coeff is loaded by `tpu_bdc_load_fp32_exp_coeff()`, and the table is loaded by `tpu_bdc_load_fp32_exp_table()`.
- The work0 and work1 tensors are workspaces to store temporary tensors. Any two of `dst_addr`, `src_addr`, `work0_addr` and `work1_addr` are not allowed to be equal. If there is no bank conflicting between any two of destination, work1 and table tensors, the performance will be better.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are in $[1, 65535]$. `shape->h * shape->w` less than or equal to 65535.
- If `src(n, c, h, w)` is less than -103 or greater than 88, then $e^{\text{src}(n,c,h,w)}$ will be e^{-103} or e^{88} , respectively.

4.18.14 tpu_bdc_fp_hsigmoid

Calculate *HARDSIGMOID* of the source tensor.

```
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{if } x \leq -3} \\
\mathsf{1} & \mathsf{\text{if } x \geq 3} \\
\mathsf{\alpha \times \left( \mathsf{src}(n, c, h, w) + \beta \right)} & \mathsf{\text{Otherwise.}} \end{cases}}
.. math:: \mathsf{\alpha = 0.16666666666666666666666666666667}
.. math:: \mathsf{\beta = 0.5}
```

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **work0_addr** – Address of the work0.
- **shape** – Pointer to the shape of the dst, src, work0.
- **alpha** – The constant value.
- **beta** – The constant value.
- **dtype** – Data type of source and destination tensors.

Remarks

- The dst, src, work0 start at the same NPU, and all are in 64-byte aligned layout.

4.18.15 tpu_bdc_fp_hswish

Calculate *HARDSWISH* of the source tensor.

```
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{if } x \leq -3} \\
\mathsf{\frac{\mathsf{src}(n, c, h, w) + 0.5}{6}} & \mathsf{\text{if } x \geq 3} \\
\mathsf{\mathsf{src}(n, c, h, w)} & \mathsf{\text{Otherwise.}} \end{cases}}
```

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **work0_addr** – Address of the work0.
- **shape** – Pointer to the shape of the dst, src, work0.
- **dtype** – Data type of source and destination tensors.

Remarks

- The dst, src, work0 start at the same NPU, and all are in 64-byte aligned layout.

4.19 Scatter and Gather Functions

4.19.1 tpu_bdc_w_gather

Gather tensor by w index, that is, $\text{output} = \text{param}[\text{index}]$.

```
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)
```

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

Parameters

- **output_addr** – Address of the output tensor.
- **param_addr** – Address of the param.
- **index_addr** – Address of the index.
- **shape** – Pointer to the shape of the output tensor.
- **param_w** – The w of the param.
- **dtype** – Data type of the output and param.
- **index_dtype** – Data type of the index.

Remarks

- The output and param tensors start at the same NPU, and are both in 64-byte aligned layout. The index starts at the NPU 0, and is in compact layout.
- The $\text{shape} \rightarrow h$ must be 1. The shape of param is $[\text{shape} \rightarrow n, \text{shape} \rightarrow c, 1, \text{param}_w]$. The shape of index is $[1, \text{shape} \rightarrow w, 1, 1]$.
- $\text{shape} \rightarrow n, \text{shape} \rightarrow c, \text{shape} \rightarrow w$ and param_w are in $[1, 65535]$.
- The valid choices of **index_dtype** are *DT_UINT16* and *DT_UINT8*. The element of index is in $[0, \text{param}_w - 1]$.

4.19.2 tpu_bdc_w_gather_exception

Gather tensor by w index, that is, $\text{output} = \text{param}[\text{index}]$. Filling a constant when the index is maximum.

```
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)
```

$$\text{output}(n, c, 0, w) = \begin{cases} \text{param}(n, c, 0, \text{index}(0, w, 0, 0)) & \text{if } \text{index}(0, w, 0, 0) \text{ is not maximum.} \\ C & \text{if } \text{index}(0, w, 0, 0) \text{ is maximum and } \text{fill_const} \text{ is true.} \end{cases}$$

Parameters

- **output_addr** – Address of the output tensor.
- **param_addr** – Address of the param.
- **index_addr** – Address of the index.
- **C** – Constant value.

- **shape** – Pointer to the shape of the output tensor.
- **param_w** – The w of the param.
- **dtype** – Data type of the output, param and `C`.
- **index_dtype** – Data type of the index.
- **fill_const** – Flag of filling a when the index is maximum.

Remarks

- The output and param tensors start at the same NPU, and are both in 64-byte aligned layout. The index starts at the NPU 0, and is in compact layout.
- The `shape->h` must be 1. The shape of param is `[shape->n, shape->c, 1, param_w]`. The shape of index is `[1, shape->w, 1, 1]`.
- `shape->n`, `shape->c`, `shape->w` and `param_w` are in `[1, 65535]`.
- The valid choices of `index_dtype` are `DT_UINT16` and `DT_UINT8`. The element of index is in `[0, param_w - 1]` and less than maximum. If `index_dtype` is `DT_UINT16`, the maximum index is 65535. If `index_dtype` is `DT_UINT8`, the maximum index is 255.
- If the index is the maximum and `fill_const` is false, the element of output will not be filled.

4.19.3 tpu_bdc_w_scatter

Scatter tensor by `w` index, that is, `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)
```

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

Parameters

- **output_addr** – Address of the output tensor.
- **param_addr** – Address of the param.
- **index_addr** – Address of the index.
- **shape** – Pointer to the shape of the output tensor.
- **param_w** – The w of the param.
- **dtype** – Data type of the output and param.
- **index_dtype** – Data type of the index.

Remarks

- The output and param tensors start at the same NPU, and are both in 64-byte aligned layout. The index starts at the NPU 0, and is in compact layout.
- The `shape->h` must be 1. The shape of param is `[shape->n, shape->c, 1, param_w]`. The shape of index is `[1, param_w, 1, 1]`.
- `shape->n`, `shape->c`, `shape->w` and `param_w` are in `[1, 65535]`.
- The valid choices of `index_dtype` are `DT_UINT16` and `DT_UINT8`. The element of index is in `[0, shape->w - 1]`.

4.19.4 tpu_bdc_hw_gather

Gather tensor by h and w indices, that is, $\text{output} = \text{param}[\text{index}]$.

```
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)
```

$$\text{output}(n, c, h, w) = \text{param}(n, c, h_{\text{param}}, w_{\text{param}})$$

$$h_{\text{param}} = \text{index}(0, h \times W_{\text{output}} + w, 0, 1)$$

$$w_{\text{param}} = \text{index}(0, h \times W_{\text{output}} + w, 0, 0)$$

Parameters

- **output_addr** – Address of the output tensor.
- **param_addr** – Address of the param.
- **index_addr** – Address of the index.
- **shape** – Pointer to the shape of the output tensor.
- **param_h** – The h of the param.
- **param_w** – The w of the param.
- **dtype** – Data type of the output and param.
- The output and param tensors start at the same NPU, and are both in 64-byte aligned layout. The index starts at the NPU 0, and is in compact layout.
- The shape of param is $[\text{shape} \rightarrow n, \text{shape} \rightarrow c, \text{param_h}, \text{param_w}]$. The shape of index is $[1, \text{shape} \rightarrow h * \text{shape} \rightarrow w, 1, 2]$.
- $\text{shape} \rightarrow n, \text{shape} \rightarrow c, \text{shape} \rightarrow h, \text{shape} \rightarrow w, \text{param_h}$ and param_w are in $[1, 65535]$. $\text{shape} \rightarrow h * \text{shape} \rightarrow w$ less than or equal to 65535.
- Data type of the index is *DT_UINT16*. The $\text{index}(0, k, 0, 0)$ stores the coordinates of the w dimension of param, the value range is $[0, \text{param_w} - 1]$. The $\text{index}(0, k, 0, 1)$ stores the coordinates of the h dimension of param, the value range is $[0, \text{param_h} - 1]$.

4.19.5 tpu_bdc_hw_gather_exception

Gather tensor by h and w indices, that is, $\text{output} = \text{param}[\text{index}]$. Filling a constant when the index is maximum.

```
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)
```

$$\text{output}(n, c, h, w) = \begin{cases} \text{param}(n, c, h_{\text{param}}, w_{\text{param}}) & \text{Neither } h_{\text{param}} \text{ or } w_{\text{param}} \text{ is maximum.} \\ C & \text{if } h_{\text{param}} \text{ or } w_{\text{param}} \text{ is maximum and fill_const is true} \end{cases}$$

$$h_{\text{param}} = \text{index}(0, h \times W_{\text{output}} + w, 0, 1)$$

$$w_{\text{param}} = \text{index}(0, h \times W_{\text{output}} + w, 0, 0)$$

Parameters

- **output_addr** – Address of the output tensor.

- **param_addr** – Address of the param.
 - **index_addr** – Address of the index.
 - **C** – Constant value.
 - **shape** – Pointer to the shape of the output tensor.
 - **param_h** – The h of the param.
 - **param_w** – The w of the param.
 - **dtype** – Data type of the output, param and **C**.
 - **fill_const** – Flag of filling a when the index is maximum.
- The output and param tensors start at the same NPU, and are both in 64-byte aligned layout. The index starts at the NPU 0, and is in compact layout.
 - The shape of param is [shape->n, shape->c, param_h, param_w]. The shape of index is [1, shape->h * shape->w, 1, 2].
 - shape->n, shape->c, shape->h, shape->w, param_h and param_w are in [1, 65535]. shape->h * shape->w less than or equal to 65535.
 - Data type of the index is **DT_UINT16**. The maximum index is 65535. The index(0, k, 0, 0) stores the coordinates of the w dimension of param, the value range is [0, param_w - 1] and less than maximum. The index(0, k, 0, 1) stores the coordinates of the h dimension of param, the value range is [0, param_h - 1] and less than maximum.
 - If the index is the maximum and fill_const is false, the element of output will not be filled.

4.19.6 tpu_bdc_hw_scatter

Scatter tensor by *h* and *w* indices, that is, $\text{output}[\text{index}] = \text{param}$.

```
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)
```

$$\text{output}(n, c, h, w) = \text{param}(n, c, h_{\text{param}}, w_{\text{param}})$$

$$h = \text{index}(0, h_{\text{param}} \times W_{\text{param}} + w_{\text{param}}, 0, 1)$$

$$w = \text{index}(0, h_{\text{param}} \times W_{\text{param}} + w_{\text{param}}, 0, 0)$$

Parameters

- **output_addr** – Address of the output tensor.
 - **param_addr** – Address of the param.
 - **index_addr** – Address of the index.
 - **shape** – Pointer to the shape of the output tensor.
 - **param_h** – The h of the param.
 - **param_w** – The w of the param.
 - **dtype** – Data type of the output and param.
- The output and param tensors start at the same NPU, and are both in 64-byte aligned layout. The index starts at the NPU 0, and is in compact layout.
 - The shape of param is [shape->n, shape->c, param_h, param_w]. The shape of index is [1, param_h * param_w, 1, 2].

- `shape->n`, `shape->c`, `shape->h`, `shape->w`, `param_h` and `param_w` are in `[1, 65535]`. `param_h * param_w` less than or equal to 65535.
- Data type of the index is `DT_UINT16`. The `index(0, k, 0, 0)` stores the coordinates of the `w` dimension of output, the value range is `[0, shape->w - 1]`. The `index(0, k, 0, 1)` stores the coordinates of the `h` dimension of output, the value range is `[0, shape->h - 1]`.

4.19.7 tpu_bdc_batch_bcast_w_gather

Gather tensor by `w` index, that is, `output = param[index]`. The batch of the `param` tensor is broadcast.

```
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)
```

$$\text{output}(n, c, 0, w) = \begin{cases} \text{param}(0, c \bmod \text{NPU_NUM}, 0, \text{index}(n, c, 0, w)) & \text{if param is repeated.} \\ \text{param}(0, c, 0, \text{index}(n, c, 0, w)) & \text{Otherwise.} \end{cases}$$

Parameters

- `output_addr` – Address of the output tensor.
- `param_addr` – Address of the param.
- `index_addr` – Address of the index.
- `shape` – Pointer to the shape of the output and index.
- `param_w` – The `w` of the param.
- `dtype` – Data type of the output and param.
- `index_dtype` – Data type of the index.
- `is_param_repeated` – Flag whether param is repeated.

Remarks

- The output, param and index start at the same NPU, and all are in 64-byte aligned layout.
- The `shape->h` must be 1. The shape of param is `[1, shape->c, 1, param_w]`.
- `shape->n`, `shape->c`, `shape->w` and `param_w` are in `[1, 65535]`.
- The valid choices of `index_dtype` are `DT_UINT16` and `DT_UINT8`. The element of index is in `[0, param_w - 1]`.
- If there is no bank conflicting between any two of output, param and index, the performance will be better. If `is_param_repeated` is true, the shape of the param will be calculated as `[1, 1, 1, param_w]` when judging conflict.

4.19.8 tpu_bdc_batch_bcast_w_gather_exception

Gather tensor by w index, that is, $\text{output} = \text{param}[\text{index}]$. The batch of the param tensor is broadcast. Filling a constant when the index is maximum.

```
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_t *shape, int param_w,
                                             data_type_t dtype, data_type_t
                                             index_dtype, bool is_param_repeated,
                                             bool fill_const)
```

$$\text{output}(n, c, 0, w) = \begin{cases} \text{param}(0, s, 0, \text{index}(n, c, 0, w)) & \text{if index}(n, c, 0, w) \text{ is not maximum.} \\ C & \text{if index}(n, c, 0, w) \text{ is maximum and fill_const is true.} \end{cases}$$

$$s = \begin{cases} c \bmod \text{NPU_NUM} & \text{if param is repeated.} \\ c & \text{Otherwise.} \end{cases}$$

Parameters

- **output_addr** – Address of the output tensor.
- **param_addr** – Address of the param.
- **index_addr** – Address of the index.
- **C** – Constant value.
- **shape** – Pointer to the shape of the output and index.
- **param_w** – The w of the param.
- **dtype** – Data type of the output, param and C .
- **index_dtype** – Data type of the index.
- **is_param_repeated** – Flag whether param is repeated.
- **fill_const** – Flag of filling a constant when the index is maximum.

Remarks

- The output, param and index start at the same NPU, and all are in 64-byte aligned layout.
- The **shape->h** must be 1. The shape of param is $[1, \text{shape->c}, 1, \text{param_w}]$.
- **shape->n**, **shape->c**, **shape->w** and **param_w** are in $[1, 65535]$.
- The valid choices of **index_dtype** are [DT_UINT16](#) and [DT_UINT8](#). The element of index is in $[0, \text{param_w} - 1]$ and less than maximum. If **index_dtype** is [DT_UINT16](#), the maximum index is 65535. If **index_dtype** is [DT_UINT8](#), the maximum index is 255.
- If the index is the maximum and **fill_const** is false, the element of output will not be filled.
- If there is no bank conflicting between any two of output, param and index, the performance will be better. If **is_param_repeated** is true, the shape of the param will be calculated as $[1, 1, 1, \text{param_w}]$ when judging conflict.

4.19.9 tpu_bdc_batch_bcast_w_scatter

Scatter tensor by w index, that is, $\text{output}[\text{index}] = \text{param}$. The batch of the param tensor is broadcast.

```
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)
```

$$\text{output}(n, c, 0, \text{index}(n, c, 0, w)) = \begin{cases} \text{param}(0, c \bmod \text{NPU_NUM}, 0, w) & \text{if param is repeated.} \\ \text{param}(0, c, 0, w) & \text{Otherwise.} \end{cases}$$

Parameters

- **output_addr** – Address of the output tensor.
- **param_addr** – Address of the param.
- **index_addr** – Address of the index.
- **shape** – Pointer to the shape of the output tensor.
- **param_w** – The w of the param.
- **dtype** – Data type of the output and param.
- **index_dtype** – Data type of the index.
- **is_param_repeated** – Flag whether param is repeated.

Remarks

- The output, param and index start at the same NPU, and all are in 64-byte aligned layout.
- The $\text{shape} \rightarrow h$ must be 1. The shape of param is $[1, \text{shape} \rightarrow c, 1, \text{param_w}]$. The shape of index is $[\text{shape} \rightarrow n, \text{shape} \rightarrow c, 1, \text{param_w}]$.
- $\text{shape} \rightarrow n$, $\text{shape} \rightarrow c$, $\text{shape} \rightarrow w$ and param_w are in $[1, 65535]$.
- The valid choices of **index_dtype** are *DT_UINT16* and *DT_UINT8*. The element of index is in $[0, \text{shape} \rightarrow w - 1]$.
- If there is no bank conflicting between any two of output, param and index, the performance will be better. If **is_param_repeated** is true, the shape of the param will be calculated as $[1, 1, 1, \text{param_w}]$ when judging conflict.

4.19.10 tpu_bdc_batch_bcast_w_mask_select

Select elements of output by the w index of mask tensor, that is, $\text{output} = \text{param}[\text{mask}]$. Meanwhile, count the number of non-zero values in the mask. The batch of the param tensor is broadcast.

```
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)
```

There are R nonzero values for fixed n and c : $\text{mask}(n, c, 0, w_0), \text{mask}(n, c, 0, w_1), \dots, \text{mask}(n, c, 0, w_{R-1})$

$$\text{output}(n, c, 0, r) = \begin{cases} \text{param}(0, c \bmod \text{NPU_NUM}, 0, w_r) & \text{if param is repeated.} \\ \text{param}(0, c, 0, w_r) & \text{Otherwise.} \end{cases}$$

$$\text{count}(n, c, 0, 0) = R$$

Parameters

- **output_addr** – Address of the output tensor.
- **count_addr** – Address of the count.
- **param_addr** – Address of the param.
- **mask_addr** – Address of the mask.
- **shape** – Pointer to the shape of the output and mask.
- **dtype** – Data type of the output and param.
- **mask_dtype** – Data type of the mask.
- **is_param_repeated** – Flag whether param is repeated.

Remarks

- The output, count, param and mask start at the same NPU. The output, param and mask are all in 64-byte aligned layout, and the count is in compact layout.
- The **shape->h** must be 1. The shape of count is [**shape->n**, **shape->c**, 1, 1]. The shape of param is [1, **shape->c**, 1, **shape->w**].
- **shape->n**, **shape->c**, **shape->w** are in [1, 65535].
- The valid choices of **mask_dtype** are *DT_UINT32*, *DT_UINT16* and *DT_UINT8*. The data type of the count is *DT_UINT16*, with the range of [0, **shape->w**].
- If there is no bank conflicting between any two of output, count, param and mask, the performance will be better. If **is_param_repeated** is true, the shape of the param will be calculated as [1, 1, 1, **shape->w**] when judging conflict.

4.19.11 tpu_bdc_batch_bcast_h_gather

Gather tensor by *h* index, that is, $\text{output} = \text{param}[\text{index}]$. The batch of the param tensor is broadcast.

```
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)
```

$$\text{output}(n, c, h, w) = \begin{cases} \text{param}(0, c \bmod \text{NPU_NUM}, \text{index}(n, c, h, 0), w) & \text{if param is repeated.} \\ \text{param}(0, c, \text{index}(n, c, h, 0), w) & \text{Otherwise.} \end{cases}$$

Parameters

- **output_addr** – Address of the output tensor.
- **param_addr** – Address of the param.
- **index_addr** – Address of the index.
- **shape** – Pointer to the shape of the output tensor.
- **param_h** – The *h* of the param.
- **dtype** – Data type of the output and param.
- **index_dtype** – Data type of the index.
- **is_param_repeated** – Flag whether param is repeated.

Remarks

- The output, param and index start at the same NPU. The output and param are both in line 64-byte aligned layout, and the index is in 64-byte aligned layout.
- The shape of param is [1, shape->c, param_h, shape->w]. The shape of index is [shape->n, shape->c, shape->h, 1].
- shape->n, shape->c, shape->h, shape->w and param_h are in [1, 65535].
- The valid choices of index_dtype are *DT_UINT16* and *DT_UINT8*. The element of index is in [0, param_h - 1].

4.19.12 tpu_bdc_batch_bcast_h_gather_exception

Gather tensor by *h* index, that is, output = param[index]. The batch of the param tensor is broadcast. Filling a constant when the index is maximum.

```
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)
```

$$\text{output}(n, c, h, w) = \begin{cases} \text{param}(0, s, \text{index}(n, c, h, 0), w) & \text{if index}(n, c, h, 0) \text{ is not maximum.} \\ C & \text{if index}(n, c, h, 0) \text{ is maximum and fill_const is true.} \end{cases}$$

$$s = \begin{cases} c \bmod \text{NPU_NUM} & \text{if param is repeated.} \\ c & \text{Otherwise.} \end{cases}$$

Parameters

- **output_addr** – Address of the output tensor.
- **param_addr** – Address of the param.
- **index_addr** – Address of the index.
- **C** – Constant value.
- **shape** – Pointer to the shape of the output tensor.
- **param_h** – The h of the param.
- **dtype** – Data type of the output, param and *C*.
- **index_dtype** – Data type of the index.
- **is_param_repeated** – Flag whether param is repeated.
- **fill_const** – Flag of filling a constant when the index is maximum.

Remarks

- The output, param and index start at the same NPU. The output and param are both in line 64-byte aligned layout, and the index is in 64-byte aligned layout.
- The shape of param is [1, shape->c, param_h, shape->w]. The shape of index is [shape->n, shape->c, shape->h, 1].
- shape->n, shape->c, shape->h, shape->w and param_h are in [1, 65535].
- The valid choices of index_dtype are *DT_UINT16* and *DT_UINT8*. The element of index is in [0, param_h - 1] and less than maximum. If index_dtype is *DT_UINT16*, the maximum index is 65535. If index_dtype is *DT_UINT8*, the maximum index is 255.

- If the index is the maximum and `fill_const` is false, the element of output will not be filled.

4.19.13 tpu_bdc_batch_bcast_h_scatter

Scatter tensor by h index, that is, $\text{output}[\text{index}] = \text{param}$. The batch of the param tensor is broadcast.

```
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)
```

$$\text{output}(n, c, \text{index}(n, c, h, 0), w) = \begin{cases} \text{param}(0, c \bmod \text{NPU_NUM}, h, w) & \text{if param is repeated.} \\ \text{param}(0, c, h, w) & \text{Otherwise.} \end{cases}$$

Parameters

- **output_addr** – Address of the output tensor.
- **param_addr** – Address of the param.
- **index_addr** – Address of the index.
- **shape** – Pointer to the shape of the output tensor.
- **param_h** – The h of the param.
- **dtype** – Data type of the output and param.
- **index_dtype** – Data type of the index.
- **is_param_repeated** – Flag whether param is repeated.

Remarks

- The output, param and index start at the same NPU. The output and param are both in line 64-byte aligned layout, and the index is in 64-byte aligned layout.
- The shape of param is $[1, \text{shape} \rightarrow c, \text{param_h}, \text{shape} \rightarrow w]$. The shape of index is $[\text{shape} \rightarrow n, \text{shape} \rightarrow c, \text{param_h}, 1]$.
- $\text{shape} \rightarrow n, \text{shape} \rightarrow c, \text{shape} \rightarrow h, \text{shape} \rightarrow w$ and param_h are in $[1, 65535]$.
- The valid choices of `index_dtype` are `DT_UINT16` and `DT_UINT8`. The element of index is in $[0, \text{param_h} - 1]$.

4.20 Special Functions

4.20.1 tpu_bdc_fp_taylor

Calculate the Taylor series of a tensor.

```
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)
```

$$\text{dst}(n, c, h, w) = \sum_{i=0}^{\text{num}-1} \text{coeff}(0, c \bmod \text{NPU_NUM}, 0, i) \times \text{src}(n, c, h, w)^i$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **coeff_addr** – Address of the coeff.
- **shape** – Pointer to the shape of the destination and source tensors.
- **num** – Number of terms in Taylor series.
- **dtype** – Data type of the dst, src and coeff.

Remarks

- The destination and source tensors start at the same NPU, and are both in 64-byte aligned layout.
- The coeff starts at NPU 0, with the shape of [1, *NPU_NUM*, 1, num], and is in 64-byte aligned layout.
- num ≥ 2.
- shape->n, shape->c, shape->h and shape->w are all in [1, 65535].

4.20.2 tpu_bdc_table_lookup

Look up the table by elements of the source tensor.

```
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)
```

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

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **table_addr** – Address of the table.
- **shape** – Pointer to the shape of the destination and source tensors.
- **len** – Length of the table.
- **dst_dtype** – Data type of the destination tensor and table.
- **src_dtype** – Data type of the source tensor.

Remarks

- The destination and source tensors start at the same NPU, and are both in 64-byte aligned layout.
- The table starts at NPU 0, with the shape of [1, *NPU_NUM*, 1, len], and is in 64-byte aligned layout.
- shape->n, shape->c, shape->h and shape->w are all in [1, 65535]. shape->h * shape->w is less than or equal to 65535.
- The valid choices of **src_dtype** are *DT_UINT16* and *DT_UINT8*. The value range of the source tensor is [0, len - 1].
- If there is no bank conflicting between any two of destination, source and table, the performance will be better.

4.20.3 tpu_bdc_arithmetic_sequence_bcast

Generate arithmetic sequence and broadcast to each NPU.

```
void tpu_bdc_arithmetic_sequence_bcast(local_addr_t dst_addr, int npu_num, int
                                         start, int step, int num)
```

$$\text{dst}(0, c, 0, w) = \text{start} + \text{step} \times w$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **npu_num** – Number of NPUs to be broadcasted.
- **start** – The start of arithmetic sequence. The start of arithmetic sequence.
- **step** – The step of arithmetic sequence.
- **num** – The number of terms in arithmetic sequence.

Remarks

- The destination tensor has the shape of [1, npu_num, 1, num] and the data type is *DT_INT32*, and is in 64-byte aligned layout.
- If the destination tensor starts at NPU X, the X is in [0, *NPU_NUM* - 1], npu_num ≤ *NPU_NUM* - X.
- num > 0.

4.20.4 tpu_bdc_arithmetic_sequence_distribute

Generate arithmetic sequence and distribute to each NPU.

```
void tpu_bdc_arithmetic_sequence_distribute(local_addr_t dst_addr, int start, int
                                              step, int num)
```

$$\text{dst}(0, c, 0, 0) = \text{start} + \text{step} \times c$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **start** – The start of arithmetic sequence.
- **step** – The step of arithmetic sequence.
- **num** – The number of terms in arithmetic sequence.

Remarks

- The destination tensor starts at NPU 0, and is in compact layout.
- The destination tensor has the shape of [1, num, 1, 1] and the data type is *DT_INT32*.
- num > 0.

4.20.5 tpu_bdc_arithmetic_sequence_general

Generate arithmetic sequence and distribute to each NPU, each step can be set independently.

```
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)
```

$$\text{dst}(0, c, 0, w) = \text{start} + \text{step} \times (\text{num} \times c + w)$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **buffer_addr** – Address of the buffer.
- **npu_num** – Number of the channels.
- **start** – The start of arithmetic sequence.
- **step** – The step of arithmetic sequence.
- **num** – The number of terms in arithmetic sequence.

Remarks

- The destination tensor has the shape of [1, **npu_num**, 1, **num**] and the data type is *DT_INT32*, and is in 64-byte aligned layout.
- The destination and buffer tensors start at NPU 0.
- The buffer tensor has the shape of [1, *NPU_NUM*, 1, 1] and the data type is *DT_INT32*, and is in compact layout.
- **npu_num** > 0.
- **num** > 0.

4.20.6 tpu_bdc_load_fp32_exp_coeff

Load the Taylor coefficients of exponential function into local memory.

```
void tpu_bdc_load_fp32_exp_coeff(local_addr_t coeff_addr)
```

Parameters **coeff_addr** – Address of the coeff.

Remarks

- The coeff starts at NPU 0, with the shape of [1, *NPU_NUM*, 1, 32], and is in 64-byte aligned layout.

4.20.7 tpu_bdc_load_fp32_exp_table

Load the table of exponent values from -103 to 88 into local memory.

```
void tpu_bdc_load_fp32_exp_table(local_addr_t table_addr)
```

Parameters **table_addr** – Address of the table.

Remarks

- The table starts at NPU 0, with the shape of [1, *NPU_NUM*, 1, 192], and is in 64-byte aligned layout.

4.20.8 tpu_bdc_load_fp32_log_coeff

Load the Taylor coefficients of the logarithmic function into local memory.

```
void tpu_bdc_load_fp32_log_coeff(local_addr_t coeff_addr)
```

Parameters `coeff_addr` – Address of the coeff.

Remarks

- The coeff starts at NPU 0, with the shape of [1, *NPU_NUM*, 1, 32], and is in 64-byte aligned layout.

4.20.9 tpu_bdc_load_fp32_erf_coeff

Load the Taylor coefficients of the error function (*erf*) into local memory.

```
void tpu_bdc_load_fp32_erf_coeff(local_addr_t coeff_addr)
```

Parameters `coeff_addr` – Address of the coeff.

Remarks

- The coeff starts at NPU 0, with the shape of [1, *NPU_NUM*, 1, 10], and is in 64-byte aligned layout.

4.20.10 tpu_bdc_load_fp32_sin_coeff

Load the Taylor coefficients of the sine function (*sin*) into local memory.

```
void tpu_bdc_load_fp32_sin_coeff(local_addr_t coeff_addr)
```

Parameters `coeff_addr` – Address of the coeff.

Remarks

- The coeff starts at NPU 0, with the shape of [1, *NPU_NUM*, 1, 32], and is in 64-byte aligned layout.

4.20.11 tpu_bdc_load_fp32_cos_coeff

Load the Taylor coefficients of the cosine function (*cos*) into local memory.

```
void tpu_bdc_load_fp32_cos_coeff(local_addr_t coeff_addr)
```

Parameters `coeff_addr` – Address of the coeff.

Remarks

- The coeff starts at NPU 0, with the shape of [1, *NPU_NUM*, 1, 32], and is in 64-byte aligned layout.

4.20.12 tpu_bdc_load_fp32_tan_coeff

Load the Taylor coefficients of the tangent function (*tan*) into local memory.

```
void tpu_bdc_load_fp32_tan_coeff(local_addr_t coeff_addr)
```

Parameters *coeff_addr* – Address of the coeff.

Remarks

- The coeff starts at NPU 0, with the shape of [1, *NPU_NUM*, 1, 32], and is in 64-byte aligned layout.

4.20.13 tpu_bdc_load_fp32_arcsin_coeff

Load the Taylor coefficients of the arcsin function into local memory.

```
void tpu_bdc_load_fp32_arcsin_coeff(local_addr_t coeff_addr)
```

Parameters *coeff_addr* – Address of the coeff.

Remarks

- The coeff starts at NPU 0, with the shape of [1, *NPU_NUM*, 1, 32], and is in 64-byte aligned layout.

4.21 Quantization Functions

4.21.1 tpu_bdc_int_requant

Requantize the tensor, and perform saturation on the result.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (\text{src}(n, c, h, w) \times \text{multiplier} \text{ LSH } \text{shift}) + \text{offset} & \text{if shift} > 0 \\ (\text{src}(n, c, h, w) \times \text{multiplier} \text{ RSH } - \text{shift}) + \text{offset} & \text{Otherwise.} \end{cases}$$

Parameters

- *dst_addr* – Address of the destination tensor.
- *src_addr* – Address of the source tensor.
- *shape* – Pointer to the shape of the destination and source tensors.
- *multiplier* – The value of multiplier.
- *shift* – Number of shifts.
- *offset* – The value of offset. The value of offset.
- *dst_dtype* – Data type of the destination tensor.
- *src_dtype* – Data type of the source tensor.
- *rounding_mode* – Rounding mode for shif

Remarks

- The destination and source tensors start at the same NPU, and are both in 64-byte aligned layout.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are all in $[1, 65535]$.
- The valid choices of `dst_dtype` are `DT_INT16`, `DT_UINT16`, `DT_INT8` and `DT_UINT8`. The valid choices of `src_dtype` are `DT_INT32`, `DT_INT16` and `DT_UINT16`. If `dst_dtype` is signed, the data type of `offset` is `DT_INT16`, otherwise it is `DT_UINT16`.
- `shift` $\in [-64, 31]$.

4.21.2 tpu_bdc_int_pc_requant

Requantize the tensor by channel, and perform saturation on the result.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (\text{src}(n, c, h, w) \times \text{quant}(0, c, 0, 0) \text{ LSH } \text{quant}(0, c, 0, 1)) + \text{quant}(0, c, 0, 2) & \text{if } \text{quant}(0, c, 0, 1) > 0 \\ (\text{src}(n, c, h, w) \times \text{quant}(0, c, 0, 0) \text{ RSH } - \text{quant}(0, c, 0, 1)) + \text{quant}(0, c, 0, 2) & \\ \text{Otherwise.} & \end{cases}$$

Parameters

- `dst_addr` – Address of the destination tensor.
- `src_addr` – Address of the source tensor.
- `quant_addr` – Address of the quant.
- `shape` – Pointer to the shape of the destination and source tensors.
- `dst_dtype` – Data type of the destination tensor.
- `src_dtype` – Data type of the source tensor.
- `rounding_mode` – Rounding mode for shift.

Remarks

- The destination, source and quant tensors start at the same NPU, and are all in 64-byte aligned layout.
- The quant has the shape of $[1, \text{shape->c}, 1, 3]$, and the type of `DT_INT32`. The `quant(0, c, 0, 0)` is the multiplier. The `quant(0, c, 0, 1)` is the shift number, with value range of $[-64, 31]$. The `quant(0, c, 0, 2)` is the offset number. The range of `quant(0, c, 0, 2)` is $[-32768, 32767]$ if `dst_dtype` is signed, otherwise $[0, 65535]$.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are all in $[1, 65535]$.
- The valid choices of `dst_dtype` are `DT_INT16`, `DT_UINT16`, `DT_INT8` and `DT_UINT8`. The valid choices of `src_dtype` are `DT_INT32`, `DT_INT16` and `DT_UINT16`.

4.21.3 tpu_bdc_fp32_requant

Requantize the tensor, and perform saturation on the result.

```
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)
```

$$\text{dst}(n, c, h, w) = \text{INT}(\text{FP32}(\text{src}(n, c, h, w)) \times \text{scale} + \text{offset})$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **shape** – Pointer to the shape of the destination and source tensors.
- **scale** – The value of multiplier.
- **offset** – The value of offset.
- **dst_dtype** – Data type of the destination tensor.
- **src_dtype** – Data type of the source tensor.
- **dst_rounding_mode** – Rounding mode for elements converting from floating-point numbers to destination tensor.
- **src_rounding_mode** – Rounding mode for converting elements from the source tensor to floating-point numbers.

Remarks

- The destination and source tensors start at the same NPU, and are both in 64-byte aligned layout.
- **shape->n**, **shape->c**, **shape->h** and **shape->w** are all in [1, 65535].
- The valid choices of **dst_dtype** are *DT_INT16*, *DT_UINT16*, *DT_INT8* and *DT_UINT8*. The valid choices of **src_dtype** are *DT_INT32*, *DT_INT16* and *DT_UINT16*.
- The valid choices of **dst_rounding_mode** are *RM_HALF_TO_EVEN*, *RM_HALF_AWAY_FROM_ZERO*, *RM_TOWARDS_ZERO*, *RM_DOWN* and *RM_UP*.

4.21.4 tpu_bdc_fp32_pc_requant

Requantize the tensor by channel, and perform saturation on the result.

```
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)
```

$$\text{dst}(n, c, h, w) = \text{INT}(\text{FP32}(\text{src}(n, c, h, w)) \times \text{quant}(0, c, 0, 0) + \text{quant}(0, c, 0, 1))$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.

- **quant_addr** – Address of the quant.
- **shape** – Pointer to the shape of the destination and source tensors.
- **dst_dtype** – Data type of the destination tensor.
- **src_dtype** – Data type of the source tensor.
- **dst_rounding_mode** – Rounding mode for elements converting from floating-point numbers to destination tensor.
- **src_rounding_mode** – Rounding mode for converting elements from the source tensor to floating-point numbers.

Remarks

- The destination, source and quant tensors start at the same NPU, and are all in 64-byte aligned layout.
- The quant has the shape of [1, shape->c, 1, 2], and the type of *DT_FP32*. The quant(0, c, 0, 0) is the multiplier. The quant(0, c, 0, 1) is the offset number.
- shape->n, shape->c, shape->h and shape->w are all in [1, 65535].
- The valid choices of **dst_dtype** are *DT_INT16*, *DT_UINT16*, *DT_INT8* and *DT_UINT8*. The valid choices of **src_dtype** are *DT_INT32*, *DT_INT16* and *DT_UINT16*.
- The valid choices of **dst_rounding_mode** are *RM_HALF_TO_EVEN*, *RM_HALF_AWAY_FROM_ZERO*, *RM_TOWARDS_ZERO*, *RM_DOWN* and *RM_UP*.

4.21.5 tpu_bdc_int_dequant

Dequantize the tensor, and perform saturation on the result.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (\text{src}(n, c, h, w) - \text{offset}) \times \text{multiplier} \text{ LSH } \text{shift} & \text{if shift} > 0 \\ (\text{src}(n, c, h, w) - \text{offset}) \times \text{multiplier} \text{ RSH } - \text{shift} & \text{Otherwise.} \end{cases}$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **shape** – Pointer to the shape of the destination and source tensors.
- **offset** – The value of offset.
- **multiplier** – The value of multiplier.
- **shift** – Number of shifts.
- **dst_dtype** – Data type of the destination tensor.
- **src_dtype** – Data type of the source tensor.
- **rounding_mode** – Rounding mode for shift.

Remarks

- The destination and source tensors start at the same NPU, and are both in 64-byte aligned layout.
- shape->n, shape->c, shape->h and shape->w are all in [1, 65535].

- The valid choices of `dst_dtype` are `DT_INT32`, `DT_INT16` and `DT_UINT16`. The valid choices of `src_dtype` are `DT_INT16`, `DT_UINT16`, `DT_INT8` and `DT_UINT8`. If `src_dtype` is signed, the data type of `offset` is `DT_INT16`, otherwise it is `DT_UINT16`.
- `shift` $\in [-64, 31]$.

4.21.6 tpu_bdc_int_pc_dequant

Dequantize the tensor by channel, and perform saturation on the result.

```
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)
```

$$\text{dst}(n, c, h, w) = \begin{cases} (\text{src}(n, c, h, w) - \text{quant}(0, c, 0, 0)) \times \text{quant}(0, c, 0, 1) \text{ LSH} & \text{quant}(0, c, 0, 2) \\ \text{if } \text{quant}(0, c, 0, 2) > 0 \\ (\text{src}(n, c, h, w) - \text{quant}(0, c, 0, 0)) \times \text{quant}(0, c, 0, 1) \text{ RSH} & - \text{quant}(0, c, 0, 2) \\ \text{Otherwise.} \end{cases}$$

Parameters

- `dst_addr` – Address of the destination tensor.
- `src_addr` – Address of the source tensor.
- `quant_addr` – Address of the quant.
- `shape` – Pointer to the shape of the destination and source tensors.
- `dst_dtype` – Data type of the destination tensor.
- `src_dtype` – Data type of the source tensor.
- `rounding_mode` – Rounding mode for shift.

Remarks

- The destination, source and quant tensors start at the same NPU, and are all in 64-byte aligned layout.
- The quant has the shape of `[1, shape->c, 1, 3]`, and the type of `DT_INT32`. The `quant(0, c, 0, 0)` is the offset number. The range of `quant(0, c, 0, 0)` is `[-32768, 32767]` if `src_dtype` is signed, otherwise `[0, 65535]`. The `quant(0, c, 0, 1)` is the multiplier. The `quant(0, c, 0, 2)` is the shift number, with value range of `[-64, 31]`.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are all in `[1, 65535]`.
- The valid choices of `dst_dtype` are `DT_INT32`, `DT_INT16` and `DT_UINT16`. The valid choices of `src_dtype` are `DT_INT16`, `DT_UINT16`, `DT_INT8` and `DT_UINT8`.

4.21.7 tpu_bdc_fp32_dequant

Dequantize the tensor.

```
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)
```

$$\text{dst}(n, c, h, w) = \text{FP32}(\text{src}(n, c, h, w) - \text{offset}) \times \text{scale}$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **shape** – Pointer to the shape of the destination and source tensors.
- **offset** – The value of offset.
- **scale** – The value of multiplier.
- **src_dtype** – The data type of the source tensor and *offset*.
- **rounding_mode** – Rounding mode for fixed-point to floating-point conversions.

Remarks

- The destination and source tensors start at the same NPU, and are both in 64-byte aligned layout.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are all in [1, 65535].
- The valid choices of `src_dtype` are *DT_INT16*, *DT_UINT16*, *DT_INT8* and *DT_UINT8*.

4.21.8 tpu_bdc_fp32_pc_dequant

Dequantize the tensor by channel.

```
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)
```

$$\text{dst}(n, c, h, w) = \text{FP32}(\text{src}(n, c, h, w) - \text{quant}(0, c, 0, 0)) \times \text{quant}(0, c, 0, 1)$$

Parameters

- **dst_addr** – Address of the destination tensor.
- **src_addr** – Address of the source tensor.
- **quant_addr** – Address of the quant.
- **shape** – Pointer to the shape of the destination and source tensors.
- **src_dtype** – Data type of the source tensor.
- **rounding_mode** – Rounding mode for fixed-point to floating-point conversions.

Remarks

- The destination, source and quant tensors start at the same NPU, and are all in 64-byte aligned layout.
- The quant has the shape of [1, `shape->c`, 1, 2], and the type of *DT_INT32* or *DT_FP32*. The `quant(0, c, 0, 0)` is the offset number and has the data type of *DT_INT32*. The value range of `quant(0, c, 0, 0)` is the same as `src_dtype`. The `quant(0, c, 0, 1)` is the multiplier and has the data type of *DT_FP32*.
- `shape->n`, `shape->c`, `shape->h` and `shape->w` are all in [1, 65535]. The valid choices of `src_dtype` are *DT_INT16*, *DT_UINT16*, *DT_INT8* and *DT_UINT8*.

4.22 HAU Functions

4.22.1 tpu_hau_sort

Sort the top K smallest or largest numbers in ascending or descending order.

```
tpu_hau_sort(system_addr_t output_addr, system_addr_t input_addr, int len, int K,
            bool descended, data_type_t dtype)
```

$$\text{output}(k) = \text{input}(i_k)$$

Ascending order:

$$\text{input}(i_0) \leq \text{input}(i_1) \leq \dots \leq \text{input}(i_{K-1}) \leq \dots \leq \text{input}(i_{\text{len}-1})$$

Descending order:

$$\text{input}(i_0) \geq \text{input}(i_1) \geq \dots \geq \text{input}(i_{K-1}) \geq \dots \geq \text{input}(i_{\text{len}-1})$$

Where $i_0, i_1, \dots, i_{\text{len}-1}$ are different from each other, which is the rearrangement of $0, 1, \dots, \text{len} - 1$.

Parameters

- **output_addr** – Address of the output.
- **input_addr** – Address of the input.
- **len** – Length of the input.
- **K** – Length of sorting.
- **descended** – Flag of descended.
- **dtype** – Data type of the output and input.

Remarks

- The **output_addr** and **input_addr** are both divisible by 4.
- The valid choices of **dtype** are *DT_FP32*, *DT_INT32* and *DT_UINT32*.
- The length of output is **len**, the first K numbers are the sorted result, $K \leq \text{len}$.

4.22.2 tpu_hau_sort_natural_index

Sort the top K smallest or largest numbers in ascending or descending order and output the sorted index. The sorting is stable. The index before sorting is the natural index $[0, \text{length} - 1]$.

```
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)
```

$$\text{output_data}(k) = \text{input}(i_k) \quad \text{output_idx}(k) = i_k$$

$$\text{if } \text{input}(i_k) = \text{input}(i_{k+1}), \text{ then } i_k < i_{k+1}$$

Ascending order:

$$\text{input}(i_0) \leq \text{input}(i_1) \leq \dots \leq \text{input}(i_{K-1}) \leq \dots \leq \text{input}(i_{\text{len}-1})$$

Descending order:

$$\text{input}(i_0) \geq \text{input}(i_1) \geq \dots \geq \text{input}(i_{K-1}) \geq \dots \geq \text{input}(i_{\text{len}-1})$$

Where $i_0, i_1, \dots, i_{\text{len}-1}$ are different from each other, which is the rearrangement of $0, 1, \dots, \text{len} - 1$.

Parameters

- **output_data_addr** – Address of the output_data.
- **output_idx_addr** – Address of the output_idx.
- **input_addr** – Address of the input.
- **len** – Length of the input.
- **K** – Length of sorting.
- **descended** – Flag of descended.
- **dtype** – Data type of the output_data and input.

Remarks

- The output_data_addr, output_idx_addr and input_addr are all divisible by 4.
- The valid choices of dtype are *DT_FP32*, *DT_INT32* and *DT_UINT32*. The data type of output_idx is *DT_INT32*.
- The length of output_data and output_idx are len, the first K numbers are the sorted result and index, $K \leq \text{len}$.

4.22.3 tpu_hau_sort_specific_index

Sort the top K smallest or largest numbers in ascending or descending order and output the sorted index. The sorting is stable. The index before sorting is the specified index.

```
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)
```

$$\text{output_data}(k) = \text{input_data}(i_k) \quad \text{output_idx}(k) = \text{input_idx}(i_k)$$

$$\text{if } \text{input_data}(i_k) = \text{input_data}(i_{k+1}), \text{ then } \text{input_idx}(i_k) \leq \text{input_idx}(i_{k+1})$$

Ascending order:

$$\text{input_data}(i_0) \leq \text{input_data}(i_1) \leq \dots \leq \text{input_data}(i_{K-1}) \leq \dots \leq \text{input_data}(i_{\text{len}-1})$$

Descending order:

$$\text{input_data}(i_0) \geq \text{input_data}(i_1) \geq \dots \geq \text{input_data}(i_{K-1}) \geq \dots \geq \text{input_data}(i_{\text{len}-1})$$

Where $i_0, i_1, \dots, i_{\text{len}-1}$ are different from each other, which is the rearrangement of $0, 1, \dots, \text{len} - 1$.

Parameters

- **output_data_addr** – Address of the output_data.
- **output_idx_addr** – Address of the output_idx.
- **input_data_addr** – Address of the input_data.
- **input_idx_addr** – Address of the input_idx.
- **len** – Length of the input_data and input_idx.
- **K** – Length of sorting.
- **descended** – Flag of descended.
- **dtype** – Data type of the output_data and input.

Remarks

- The output_data_addr, output_idx_addr, input_data_addr and input_idx_addr are all divisible by 4.
- The valid choices of dtype are *DT_FP32*, *DT_INT32* and *DT_UINT32*. The data type of output_idx and input_idx are *DT_INT32*.
- The length of output_data and output_idx are len, the first K numbers are the sorted result and index, $K \leq \text{len}$.

4.22.4 tpu_hau_line_gather

Gather tensor by taking the index of the line, that is, $\text{output} = \text{param}[\text{index}]$.

```
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)
```

$$\text{output}(h, w) = \begin{cases} \text{param}(\text{index}(h) - \text{start}, w) & \text{if index}(h) \text{ is valid index,} \\ C & \text{if index}(h) \text{ is invalid index and fill_const is true.} \end{cases}$$

Parameters

- **output_addr** – Address of the output.
- **param_addr** – Address of the param.
- **index_addr** – Address of the index.
- **C** – Constant value.
- **line_num** – The number of the line of param.
- **line_len** – The length of the line of param.
- **index_len** – The length of the index.
- **start** – The start of the valid index.
- **end** – The end of the valid index.
- **dtype** – Data type of the output and param.
- **fill_const** – Flag of filling constant when index is maximum.

Remarks

- output_addr, param_addr and index_addr are all divisible by 64.

- The shape of output is `[index_len, line_len]`. The shape of param is `[line_num, line_len]`. The shape of index is `[index_len]`. All of them are in continuous layout.
- The data type of the index is `DT_INT32`, with the range of `[start, end]`. The `start` is in `[0, end]`, `end < line_num`.
- If the index is invalid and `fill_const` is false, the element of output will not be filled.