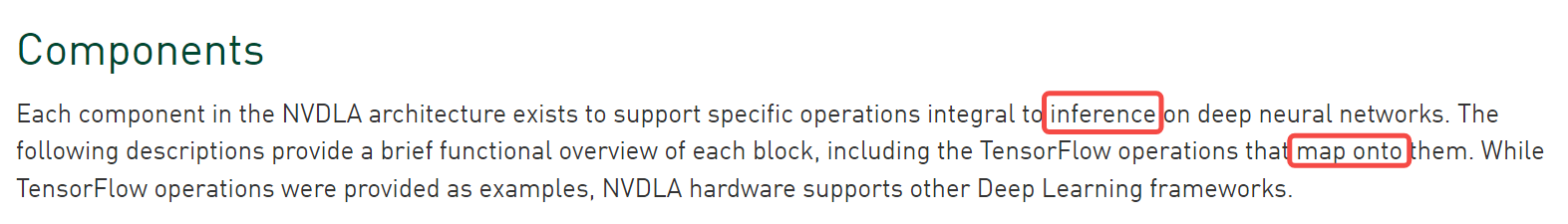
# NVDLA微架构

* 职能：推理
* 本质：实现对包括TensorFlow在内的深度学习框架的硬件映射





## Working Mode

### Independent Mode

Independent operation refers to each individual block being configured for **when and what** it executes, with each block working on its assigned task.



### Fused Mode (Improves performance)



some blocks can be assembled as a pipeline; this improves performance by bypassing the round trip through memory, instead having blocks communicate with each other through small FIFOs

##### Improving Compute Efficiency for Certain Sizes of Filters

* Built-in Winograd convolution:
  + 局限性：只适用于较小的卷积核和tile
  + 原因：Winograd卷积通过减少乘法器的数目来实现提速，代价有两方面：1）加法器、转置矩阵的计算/存储的量会增加；2）tile越大，转置矩阵越大，计算精度的丢失进一步增加

<https://www.cnblogs.com/shine-lee/p/10906535.html>

## Five-stage Convolution Pipeline

图示, 箱线图

描述已自动生成

Fig. convolution pipeline

* Convolution DMA (CDMA)
* Convolution Buffer (CBUF)
* Convolution Sequence Controller (CSC)
* Convolution MAC (CMAC)
* Convolution Accumulator (CACC)

### CDMA

* 数据加载
  + kernel: 使用Weight部件产生访存请求，内部压缩之后，将{compressed weight, WMB, WGS}通过3个独立的通道写入CBUF
  + data: 输入data先存入shared buffer，然后根据不同的卷积模式{Image Input, Winograd/Direct Conv.}对data进行“整形”后，写入CBUF
* 性能考虑
  + Winograd convolution: 为了提升直接卷积的性能。
    - 原理：通过减少乘法器的数目来实现提速

将二维卷积记为，输出为，卷积核为，则乘法数量至少为：

采用Winograd卷积，则乘法量可以较少至：

或者

<https://www.cnblogs.com/shine-lee/p/10906535.html>

* + - 代价
      * 加法器、转置矩阵的计算/存储的量会增加

e.g., a 3x3 filter-sized convolution with Winograd, reduces the number of MAC operations by a factor of 2.25x

<http://nvdla.org/hw/v1/hwarch.html>

* + - * tile越大，转置矩阵越大，计算精度的丢失进一步增加
    - 局限性
      * 只适用于较小的卷积核和tile

<https://www.cnblogs.com/shine-lee/p/10906535.html>



#### Weight Compression

* Weight mask bit (WMB): 对原始的weight按照group压缩，单个weight group被称作WMB
* Weight group size (WGS): 每一个compressed group中剩余的byte
* Weight compression组件与CBUF之间为{compressed weight, WMB, WGS}分配独立的memory interface.

weight compression组件负责对输入权重数据进行压缩，有3组输出信号与CBUF连接，具体如下：

* 抽取原始输入的uncompressed weights序列中全部的非零weight，得到第一组接口：compressed weights
* 开辟bit tags对原始的weights按照group为粒度进行表示。假设每个权重数据是int8类型，则用1-bit的tag表示kernel中对应位置的weight是否为零，1表示weight data非零，0表示weight data为零。如下图所示：原始的每组的权重数据被映射成了orig\_bit\_tag\_seq\_i。对所有group的tag bits序列进行拼接之后，按照128-byte对齐，得到最终的WMB interface
* 统计各组非零元素的数目byte\_num\_group\_i，同样基于128-Byte对齐的原则，计算每一组的WGS数值：128-byte\_num\_group\_i，并且每一个WGS采用32-bit表示。{WGS\_group\_0[31:0], WGS\_group\_1[31:0], …, WGS\_group\_n[31:0]}为第三组接口信号WGS interface.



#### Convertor

文本, 日程表

描述已自动生成

### CBUF



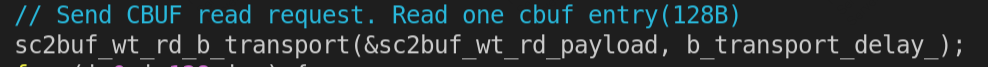
* CBUF是一个slave
  + CDMA负责对其进行填充、CDMA会**尽可能地**将kernel或者data加载到CBUF中，以避免重复地从memory中读取。
  + CBUF的数据输出受后级CSC控制
* CBUF内部的存储缓冲被data/kernel共享
* Catches input pixel/feature data, and weights.
* Resources
  + CBUF contains 16x32KB banks. Each bank has two 512-bit wide, 256-entry two-port SRAMs.



These buffers act as 3 circular buffers:

* + - Pixel/data buffer
    - Weight buffer
    - WMB
* Access Pattern
  + 对CBUF的数据请求的粒度是特定Bank的2个SRAM各自的512-bit，即1K-bit=128\*8-bit的数据量。

CModel实现的时候，以Byte为粒度进行数据搬运，从而Entry Size=1K/8=128.

本地开辟了uint8\_t\*的缓冲区cbuf\_ram\_。每次memcpy的数据量是128\*8-bit的数据。

* Usage
  + If the weight format is compressed, then bank\_15 is assigned for WMB, while bank\_0~bank\_14 can be used for the other buffers.
  + If the weight format is not compressed, then WMB is not assigned with any bank. All 16 banks can be used for pixel/data buffer and weight buffer.
  + If the total required banks are less than 16, then the remaining banks are unused.

### CSC



Fig.1 overview

* Working Flow
  + CSC poll for enough pixel/feature data and weights in each Bank in CBUF.
  + CSC\_SG generates <weight, data> pair packages. Each package represents one **stripe** operation.
  + Push data package into CSC\_DL, and weight package into CSC\_WL.
  + Both CSC\_DL and CSC\_WL own one private counter. Sync. <weight, data> pairs before flowing into CMAC.
  + Stall
    - no available DL/WL entries.
    - Either weight or data is not ready.
  + Back pressure: when the counters reach zero, check if there were free entry in CACC.
* Sequence Generator (CSC\_SG). contains the logic to execute the feature/pixel loading sequence
* Data Loader (CSC\_DL): contains the logic to execute the feature/pixel loading sequence.
  + Main Path
    - receives packages from CSC\_SG
    - loads feature/pixel data from CBUF
    - sends feature/pixel data to CMAC.
  + Control Logic
    - Maintains **data buffer status**, and communicate with CDMA\_STATUS through CSC\_SG to keep the status up to date.
  + Optional Logic
    - For Winograd convolution mode, it also performs PRA (pre-addition) to **transform** the input feature data.
* Weight Loader (CSC\_WL): contains the logic to execute the weight loading sequence
  + Main path
    - receives packages from the CSC\_SG
    - loads weights from CBUF
    - **does necessary decompression**
    - sends weight data to CMAC
  + Control Logic
    - maintains the **weight buffer status** and communicates with CDMA\_WT to keep the status up to date.

### CMAC

CMAC包含16个CMAC\_CELL



CSC访问CBUF的最小数据量是一个Bank的2个SRAM，每个SRAM的一个Entry为512-bit，即总共2\*512=1K-bit的数据量。从而，为了匹配吞吐率，CMAC内部开辟的每一个CELL涵盖64个fp16/int16的MAC单元，或者128个int8的MAC单元。

由于CBUF总共包含16个Bank，CMAC中相应地开辟了16个CELL。

每一个CMAC\_CELL包含64个16-bit（fp16/int16）的乘法器，即支持<weight[1023:0], data[1023:0]> pair的输入，如下图所示：



### CACC

The Convolution Accumulator (CACC) is the stage of the convolution pipeline after CMAC. It is used to

1. Accumulate partial sums from Convolution MAC by Assembly SRAM group
2. Round/saturate the result before sending to SDP by Delivery SRAM group.

The components in CACC include assembly SRAM group, delivery SRAM group, adder array, truncating array, valid-credit (VC) controller and a checker.

图示, 示意图

描述已自动生成

Here is the CACC working flow:

1. Prefetch accumulative sums from the assembly SRAM group.
2. When partial sums arrive, send them to adder array along with accumulative sums. If the partial sums are from the first stripe operation, the accumulative sums should be 0.
3. Gather new accumulative sums from output side of adder array.
4. Store into assembly SRAM group
5. Repeat step1~ step3 in terms of stripe operation **until a channel operation is done.**
6. If a channel operation is done, the output of adders is rounded and saturated.
7. Gather results of previous step and store them into delivery SRAM group.
8. Load results from delivery buffer group and send them to SDP

The assembly SRAM group contains 4 96Bx32 SRAMs and 4 64Bx32 SRAMs. The assembly SRAM group is used to cache accumulative sums with **high precision**.

The assembly SRAM acts differently according to convolution mode configured:

|  |  |
| --- | --- |
| Convolution mode | Resource |
| Direct convolution (DC) | * one 96B x 128 buffer for int16/fp16 * one 136B x 128 buffer for int8 |
| Winograd convolution (WC) | * one 384Bx32 buffer for int16/fp16 * one 544Bx32 buffer for int8 |

It takes at least **11 cycles** to do **a read-store** circle for assembly group.

The delivery SRAM group contains 8 64Bx32 SRAMs. The input varies from 16 elements to 128 elements per cycle, while the output is always 16 elements per cycle.

## Single Data Point Processor (SDP)

### Format conversion

SDP performs **post processing** (after conv.) at the single data element level. In NVDLA version 1.0, point processing is designed to accomplish following operations.

* Bias Addition

is the input data can either from conv. Pipe or SDP M-RDMA.

*bias* is a pre-trained parameter that can be one of 3 options:

* 1. Register: if bias is **unique** for entire data cube.
  2. SDP B/N/E-RDMA per **channel** mode: if bias is **shared** for all elements in the same channel.
  3. SDP B/N/E-RDMA per element mode: if bias is different element-by-element.
* Non-linear Function

Non-linear function in SDP is to accomplish **activation** **layer** operations.

Three activation functions are commonly used:

* 1. ReLU. For an input *x*, the output is *max*(*x*, 0)
  2. Sigmoid. For in input *x*, the output is

图表

描述已自动生成

* 1. Hyperbolic tangent (tanh), for an input *x*, the output is

图表, 折线图

描述已自动生成

ReLU can be implemented by HW logic. Sigmoid and tanh are expected to be implemented through a **LUT**.

* Batch Normalization (Layer)

Batch normalization a is widely used layer. It can be descripted by formula below

Where is the mean and is the standard variance obtained from **training**, and is element of feature data cubes.

If SDP sources data offline, the supported conversion in one hardware layer is listed in Table I

表格

描述已自动生成

Table I – Precision conversion for SDP (offline)

If SDP sources data from the **convolution core**, the supported format conversion is listed in Table II.

表格

描述已自动生成

Table II – Precision conversion for convolution layer

### HW Architecture

图示

描述已自动生成

Single point Data Processing block diagram

### Function Description

* Working mode
  + On-flying: source data is from Conv-Core
  + Off-flying: source data is from Memory, which is read by M-RDMA
* Operations
  + Bias Addition/Batch Normalization/Element-wise
    - Common

Operand data can be **per element, per channel, or per cube**. The actual operation can be performed at **any of X1/X2/Y** based on software configuration.

1. Bias data will be fetched from Memory if per element/channel. If truncate is enabled, all elements **shares a same truncate value**
2. Bias data will be set by reg. if per cube.
   * + Specific
       - For Bias Addition, MUL will be bypassed.
       - For Batch Normalization, ReLU can be bypassed or enabled.
       - For Element-wise, operand data should be either for max/min/sum, or for MUL. LUT can be bypassed or enabled.



A combination of Linear and Non-linear functions implementation.

SDP部件的吞吐量可以配置，其典型值是X模块吞吐量为每拍16个数，Y模块的吞吐量为每拍4个元素。

SDP部件的输出有两个方向，可以将结果写回存储器中，也可以直接转发至PDP部件，继续进行在线处理。

## Planar Data Processor (PDP)

The Planar Data Processor (PDP) executes operations **along the width x height plane**. In NVDLA version 1.0, the PDPD is designed to accomplish **pooling layers**. Max, min, and mean pooling methods are supported.

Several neighboring input elements within a plane will be sent to a **non-linear (**average pooling**)** function to compute one output element.

The following diagram shows an example for max-pooling. The maximum value among 3x2 neighboring elements is the pooling result value.

表格

描述已自动生成

The following diagram shows the internal blocks of the PDP sub-unit, along with connections to other units and sub-units.

PDP receives data from SDP **or** MCIF/SRAMIF, and sends data to MCIF/SRAMIF.

图示

描述已自动生成

Pooling operations are done within a plane. There is no interference between different planes.

The offset between two neighbor kernels is called stride. Let Stride\_W and Stride\_H be the strides along width direction and height direction respectively.

In the example below, pooling kernel is 5x5, Stride\_H=2, and Stride\_W=3.

Thus, there are overlaps between neighbor kernels. As indicated by the dashed arrows, the 3rd row is shared by the 1st and the 2nd kernel. Likewise, the 5th row is shared by the first 3 kernels.



If an input data element is the first element of a kernel, it will be stored to the share line buffer. Data in the share line buffer is referred to as the partial result.

If an input data element is neither the first element nor the last element of a kernel, it will be operated on with the existed partial result from share buffer, and the result will be stored to the **same entry** of the original partial result. Partial result calculation is done in the pre-processing block.

1. In cases of max/min pooling schemes, the partial result is the maximum/minimum value of the input element and the original partial result.
2. In case of **mean** pooling scheme, the partial result is the **sum** of the input element and the original partial result.

### Post-processing

Post-processing is specific for average-pooling. If the input data is the last element in a kernel, it will be operated with the existed partial result from the share line buffer to generate a pre-final result. The post-processing block will fetch pre-final results from Share Line Buffer, generate the final result, and then send out to MCIF/SRAMIF.

1. In cases of max/min pooling schemes, just side-band the pre-final results.
2. In cases of mean pooling scheme, the final result can be calculated by

Division is expensive for HW implementation, so a pair of scale factors are used to transform division into multiplication.

### Buffer Size Estimation

There are three major buffers in PDP sub-unit: Share Line Buffer, read DMA buffer, and write DMA buffer.

#### Share Line Buffer

Size of Share Line Buffer determines whether PDP could work **directly** on data from SDP or not, and it can be calculated as follow:

All the parameters are defined as below:

|  |  |
| --- | --- |
| Parameters | Definition |
| *Heightpooling\_kernel* | Pooling kernel height |
| *Stridepooling\_kernel* | Pooling kernel stride **in height direction** |
| *Groupsize* | Group size (16 elements of int16/FP16 or 32 elements of int8, ~32 byte) |
| *bytes\_per\_element* | Bytes per element (14/8 for INT8, 28/8 for INT16, 28/8 for FP16) |

If the Share Line Buffer capacity is **less than** the required consumption size, PDP have to work in **off-fly** mode, so there will be a performance drop since extra-time is needed to store data to MC/SRAM, and then fetch back to PDP for pooling processing.

#### Read DMA Buffer

For read DMA buffer, there are two constraints for determining its size. One is covering MC accessing latency, assumed to be 128 cycles. The other is access bandwidth. The peak performance case is 8 Bytes per cycle (8 elements in int8, 4 elements in int16/fp16).

So the read DMA buffer size is 128×8=1KBytes.

## Cross-channel Data Processor (CDP)

A specialized unit built to apply the local response normalization (LRN)

Cross Channel Data Processor (CDP) executes operations **along channel direction**. In NVDLA version 1.0, channel processing is designed to address local response normalization (LRN) layers. LRN performs a kind of lateral inhibition by normalizing over local input region along the channel direction. The normalization function is shown as follow

文本

描述已自动生成

Local region shape is always 1×1×*n*. Number *n* is configurable, and its range is [3,5,7,9].

CDP can be treated as a standalone lookup table (LUT) function. The LUT approach is adopted for the RESMO (reciprocation-exponent-sum-multi operation)

图表, 折线图

描述已自动生成

The following diagram shows internal blocks of the CDP sub-unit and connections to other sub-units.

图示

描述已自动生成

## Reshape (RUBIK)

RUBIK module is similar to BDMA. It transforms data mapping format without any data calculation. RUBIK has 3 working modes, they are

* contract data cube
* **split** feature data cube into multi-planar formats
* **merge** multi-planar formats to data cube

Since the module’s function is to transform feature data cubes, we call it RUBIK unit.



### Split and Merge

Split and merge are two opposite operation modes in RUBIK.

* Split transforms a data cube into M-planar formats (NCHW). The number of planes is equal to channel size.
* The merge mode transforms a serial of planes to a feature data cube.

The transform is showed in figure below.

图示

描述已自动生成

## MCIF

MCIF is used to **arbitrate** requests from several internal sub modules and **convert to AXI** protocol to connect to **external DRAM**.



MCIF will support both a read and write channels, but some NVDLA sub-module will only have read requirement, so the interface between sub-module and MCIF will support read, write or both. CDMA0 and CDMA1 in the above diagram will need read only, and other 5 will need both read and write.

## SRAMIF

The SRAMIF module is used to connect several internal sub-modules to **on-chip SRAM**. It’s **similar to the MCIF** but the bus latency is expected to be lower.



SRAMIF will support both read and write channels, but some NVDLA sub-modules will only have a read requirement, so the interface between DMA engines and SRAMIF will support read, write or both. CMDA0~1 will need read channel only, while the other 5 will need both read and write.

## Bridge DMA (BDMA)

Input images and processed results are stored in external DRAM, but external DRAM bandwidth and latency are generally insufficient to allow NVDLA to fully utilize it’s MAC arrays. Therefore NVDLA is configured with a secondary memory interface to on-chip SRAM.

To utilize the on-chip SRAM, NVDLA needs to move data between external DRAM and SRAM. Bridge DMA is proposed to full-fill this purpose.

There are two **independent** paths, one is copies data from external DRAM to internal SRAM , and the other one is copies data from internal SRAM to external DRAM. Both directions **cannot work simultaneously**.

BDMA can also move data from external DRAM to external DRAM, or from internal SRAM to internal SRAM.



## Area

图示

描述已自动生成

# 附件

## Stripe operation

图示

描述已自动生成

During one stripe operation the **weight data** in MAC cell array is kept **unchanged**. Input data slides along input data cube.