Precision Preservation

Though most software-based DNN implementations are FP32-based, many studies have already shown that lower precision is sufficient for inference. As such, NVDLA chooses to support INT8/INT16/FP16 as a trade-off between precision and performance/area. At the same time, NVDLA adopts technologies to keep the precision loss under control. Below, we give a diagram with an overview of NVDLA's precision-preservation architecture.

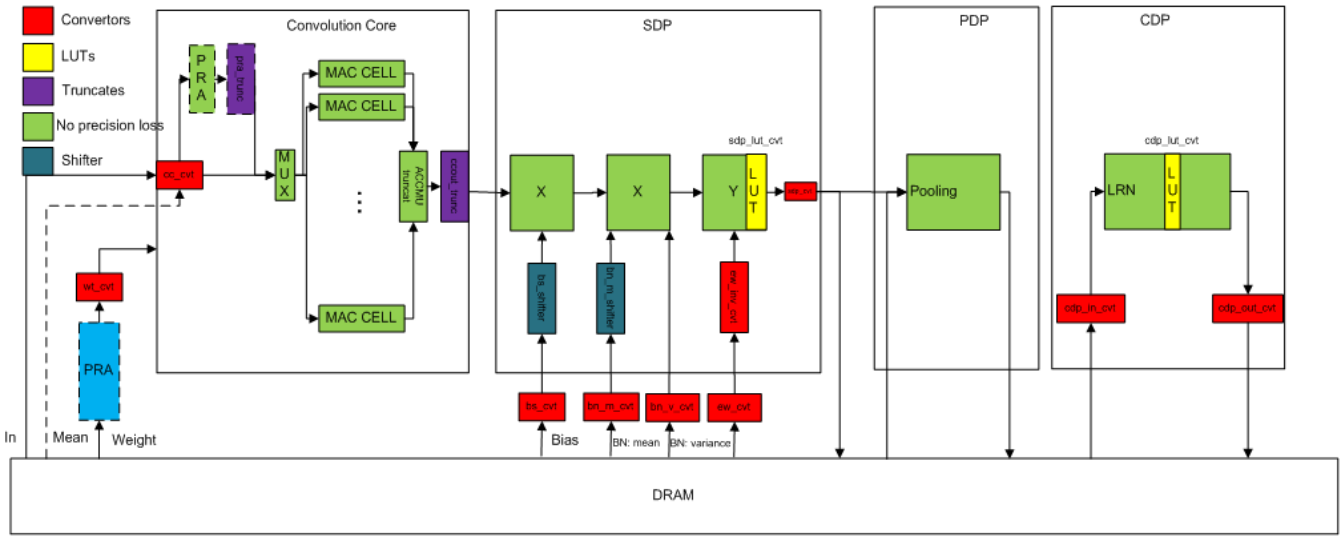

Fig. 32 - NVDLA precision-preservation architecture

In total, there are four types of approaches to precision control in the NVDLA pipeline:

- Convertor:

  The formula for a convertor in INT8 and INT16 is:
  $y = saturation\_round(x - offset_{int}) * scaling_{int} >> shifter_{uint}$

  offset, *scaling*, and *shifter* are programmable registers to allow software to control the output dynamic range. Saturation is dependent on the number of output bits.

  For INT8 and INT16, *offset* and *scaling* are treated as signed integers, and the exact number of bits is depends on the input operands. *shifter* is a 5 bits unsigned integer (always specifying a right shift); the rounding method used after the shift is to "round half away from zero".

  For FP16, the dynamic range that can be represented by FP16 representable is large, and so convertor and shifter logic is not implemented in hardware.

  The convertor is able to keep the best possible precision even if input data are not symmetric to 0, or dynamic range is not 2^N; NVDLA uses it to convert internal precision (high) to external (low, typically INT8/INT16/FP16).
- Truncate:

  Truncation is enabled for INT8/INT16 only. The formula for truncation is: $y = saturate\_round(x[msb:lsb])$

  lsb is a programmable register; *msb* is defined as *lsb* + *output_bits*.

  Similarly to the convertor, the rounding method used in truncation is also "round half away from zero".

  Truncation is used in NVDLA internal pipe where the y has enough bits (compared to convertor case); it is the result of a trade-off between precision and area.
- Shifter:

The shifter exists to make sure that the bias can be added to convolution results; it has the formula $y = saturate(x << shifter)$.

shifter is a programmable register. The saturate function depends on the number of output bits.

- LUT:

  Look-up tables are used to deal with non-linear function in networks; these functions include sigmoid/tanh activation, or local response normalization as mentioned in Data Formats (http://nvdla.org/hw/format.html). We use an innovative 2 level hybrid LUT to mimic those non-linear functions; for more detail, see LUT programming (lut-programming.html).

FP16 error threshold

We know that the output of computations on floating-point data are highly depending on computation order. As a result, when comparing the results of tests that are computed in FP16 space, we should allow some certain threshold of error when comparing NVDLA's output against reference. Below is a summary of the thresholds that are applied for each module.

Sub-module	Threshold
CC DC	$fabs(a-b) \leq 2^{max\_exp-20} * R * S * C * 2 * scale + 2^{exp(a)-10}$
CC Winograd	$fabs(a-b) \leq 2^{max\_exp-20} * 18 * C * 2 * scale + 2^{exp(a)-10}$
SDP	Bit-by-bit identical
CDP	$(fabs(a-b) \leq 0.0001) \&\& (\frac{fabs(a-b)}{max\_value} \leq 0.001)$
PDP	$(fabs(a-b) \leq 0.0001) \&\& (\frac{fabs(a-b)}{max\_value} \leq 0.001)$

In the above table, note that:

- $exp(a)$ in DC/Winograd is the operation to extract exponential field of an input: $exp(a) = ((a >> 10) \& 0x1f) - 15$.
- $max\_exp : in DC/Winograd is the maximum exponential value of : math :$ inside a convolution kernel. As data moves in a sliding window, max_exp is different element-by-element:

$$max\_exp = \max_{\substack{s=0...S-1 \\ r=0...R-1 \\ c=0...C-1}} (exp(data_{r,s,c}) \& (\sim\ 0x3) + exp(wt_{r,s,c}) \& (\sim\ 0x3))$$

- **max_value** in CDP is the maximum value inside one square sum window.
- **max_value** in PDP is the maximum value inside one pooling window.

Convertor programming

Programming interface

As mentioned above, for a given convertor, there are 3 parameters: *offset*, *scaling* and *shifter*. Depending on the use cases, those parameters have different encodings:

Parameter	INT->INT	INT->FP16	FP16->INT	FP16->FP16
Offset	INT32	INT32	Not supported	Not supported
Scaling	INT16	INT16	Not supported	Not supported
Shifter	UINT5	UINT5	Not supported	Not supported

Convolution convertors

The following is a list of place where convertors or truncation might be used for a convolution layer (please refer to Fig. 32 about where're the convertors in system):

Convertor	Functionality	Limitation	Owner

Convertor	Functionality	Limitation	Owner
cc_cvt	For image input, this convertor responsible for mean subtraction and 8 bit conversion; For feature input, this convertor won't be used	INT8/INT16 only For FP16, subtract mean data only	HW
wt_cvt	Convert weight data to INT8/16/FP16 representable	Offset is not allowed	SW
pra_trunc	Truncate the winograd pre-transformed results to INT8/16/FP16 representable	Used for winograd mode and CSC.PROC_PRECISION=INT8/INT16 only	HW
cc_out_trunc	Truncate the data to INT32/FP32 before sending to SDP	CACC.PROC_PRECISION=INT8/INT16 only	HW
bs_cvt	Convert bias data to INT8/16/FP16 representable	N/A	SW
bs_shifter	Shifter the input bias to make it addable with convolution pipeline results	SDP.PROC_PRECISION=INT8/INT16 only	HW

$$\begin{cases} SF_{bs} * 2^{bs\_trunc} = \frac{SF_{in} * SF_{wt}}{2^{pra\_trunc + cc\_out\_trunc}} & \text{conv=winograd} \\ SF_{bs} * 2^{bs\_trunc} = \frac{SF_{in} * SF_{wt}}{2^{cc\_out\_trunc}} & \text{conv=DC} \end{cases}$$

In case of input data encoded with offset (*x'=(x-offset)*SF*), this offset should be carefully considered for cases below:

- Padding:

Convolution supports zero-padding, however, if the input encoded with "offset", it means x=0 becomes x'=(0-offset)*SF=-offset*SF thus hardware should do "valued padding" instead of "zero padding". Convolution has a register named as:

PADDING_VALUE, this register should be set as –offset*SF for INT8/16 pipeline. However, for FP16, we assume there's no offset thus PADDING_VALUE should be set as 0;

- Activation:

As discussed above, activation such as ReLU is a piece wise function:

$$\begin{cases} y = x, & \text{x>0} \\ y = 0, & \text{otherwise} \end{cases}$$

0 plays a important role to decide activation output, unfortunately, if "offset" is enabled on input convolution data, the "0" is no longer 0 in the encoded activation data:

Let's deduce the CC output (activation layer input) offset based on convolution definition:

Given: $In_{int} = SF_{in} * (In - Offset_{in})$, $Wt_{int} = SF_{wt} * Wt$, $CC_{FP} = \sum In * Wt$,

$$
\begin{aligned}
CC_{int} &= \frac{\sum In_{int} * Wt_{int}}{2^{pra\_trunc+cc\_out\_trunc}} \\
&= \frac{\sum SF_{in} * (In - offset_{in}) * SF_{wt} * Wt}{2^{pra\_trunc+cc\_out\_trunc}} \\
&= \frac{SF_{in} * SF_{wt} * \sum (In - offset_{in}) * Wt}{2^{pra\_trunc+cc\_out\_trunc}} \\
&= \frac{SF_{in} * SF_{wt} * (\sum In * wt - offset_{in} * \sum Wt)}{2^{pra\_trunc+cc\_out\_trunc}} \\
&= \frac{SF_{in} * SF_{wt} * (CC_{FP} - offset_{in} * \sum Wt)}{2^{pra\_trunc+cc\_out\_trunc}}
\end{aligned}
$$

(The truncate for activation/weight are merged to SF_{in}, SF_{wt} in formula above to simplify deduction)

So, the CC output offset is: $\frac{SF_{in}*SF_{wt}*offset_{in}*\Sigma W_t}{2^{pra\_trunc+cc\_out\_trunc}}$

Please be noticed: The formula above is assuming no quantization error, in practice, there'll be quantization error on weight thus actual offset is $\frac{SF_{in}*SF_{wt}*offset_{in}*\Sigma W_t^{'}}{2^{pra\_trunc+cc\_out\_trunc}}$

Where $W_t^{'}$ is the low precision version of weight which takes weight quantization error into consideration.

$\Sigma W_t^{'}$ is different channel-by-channel which means $\frac{SF_{in}*SF_{wt}*offset_{in}*\Sigma W_t^{'}}{2^{pra\_trunc+cc\_out\_trunc}}$ also vary channel by channel thus per-channel operation has to be adopted to compensate the CC output offset. This compensation is done by ALU module in X1/X2/Y in SDP.

SDP convertors

SDP has kinds of use scenarios, table below lists how those use scenarios maps to SDP sub-modules (For the meaning of X/Y, please refer to Fig. 32)

Use scenario	Sub-module
Bias addition	X or Y
Batch Normalization	X or Y
Element-wise	X or Y
Activation(ReLU/PReLU)	X or Y
Activation(Sigmoid/TanH, etc)	Y
Precision conversion	X or Y

Let's review those cases one by one:

Bias addition

This already covered by Convolution convertors

Batch normalization

Here's a list of convertor/shifters needed to realize batch normalization function in SDP:

Convertor	Functionality	Limitation	Owner
bn_m_cvt	Convert the offline trained batch normalization mean data to INT8/16/FP16 representable	N/A	SW
bn_m_shifter	Shift the bn_m_cvt converted values to have the same scaling factor as input	For SDP.PROC_PRECISION=INT8/INT16 only	HW

Convertor	Functionality	Limitation	Owner
bn_v_cvt	Convert the offline trained batch normalization 1/variance to INT8/16/FP16 representable	Offset is not allowed	SW

The input of batch normalization should be either from CONV/MC or previous pipeline stages thus we should assume O_{in}, SF_{in} are applied on input.

In order to make mean addable with input data, formula below should be satisfied:

$$SF_{in} = SF_{bs\_m\_cvt} * 2^{bn\_m\_shifter}$$

Element wise

Here's a list of convertor/shifters needed to related to element wise operation in SDP:

Convertor	Functionality	Limitation	Owner
ew_cvt	The convertor applied on element-wise input, as element-wise are cube-based, the element-wise hardware layer are the output of upstream hardware layers	For SDP.PROC_PRECISION=INT8/INT16 only	HW
ew_inv_cvt	Align the offset/scaling factors to meet the requirement of different element wise operation(see below). If the requirement already satisfied, this convertor can be bypassed.	For SDP.PROC_PRECISION=INT8/INT16 only	HW

Since there might be 2 convertors applied on E-RDMA stream, if original input is x, the output from ew_inv_cvt is:

$$x' = \{(x - O_{ew\_cvt}) * SF_{ew\_cvt} - O_{ew\_inv\_cvt}\} * SF_{ew\_inv\_cvt} = \{x - (O_{ew\_cvt} + \frac{O_{ew\_inv\_cvt}}{SF_{ew\_cvt}})\} * SF_{ew\_cvt} * SF_{ew\_inv\_cvt}$$

In order to make element-wise acts as we supposed, the convertor parameter should be carefully configured based on different element-wise operation (Assume convertor parameter from BN module is: O_{in}, SF_{in}):

- MAX

  The offset/scaling applied on input stream and E-RDMA stream should be the same, which means:

  $$O_{in} == O_{ew\_cvt} + \frac{O_{ew\_inv\_cvt}}{SF_{ew\_cvt}}$$

  $$SF_{in} == SF_{ew\_cvt} * SF_{ew\_inv\_cvt}$$

- SUM

  The scaling factor applied on both stream should be the same:

  $$SF_{in} == SF_{ew\_cvt} * SF_{ew\_inv\_cvt}$$

- PROD

  The offset applied on E-RDMA stream should be 0:

  $$O_{ew\_cvt} + \frac{O_{ew\_inv\_cvt}}{SF_{ew\_cvt}} == 0$$

Activation (ReLU/PReLU)

The input offset of ReLU, PReLU already eliminated in ALU unit of X1/X2/Y thus the 0s in ReLU/PReLU is real "0", so, we don't need to worry modules;

Activation (Sigmoid/TanH, etc.)

If complex activation function (e.g.: sigmoid or TanH) are used, LUT has to be used to mimic the curve of those functions. The LUT coverage has to be precisely matched with the input convertor parameter to make it acts as you want.

Let's use an example to explain this match process: suppose [100, 300] is the most interesting data range, user will program LUT (suppose we have 257 LUT entries) as:

LUT[0]=f(100),

LUT[1]=f(100+200/256)

...

LUT[256]=f(300)

This means, if you want to get the correct LUT output, the LUT input has to be $x^{'} = (x - O) * SF$, where O=100, SF=200/256

So, software has to carefully program the convertors before LUT to achieve this.

Precision conversion

SDP supports various format conversions, when conversion from high precision to low (e.g.: INT16->INT8, FP16->INT16/8), a convertor is suggested to avoid the interested data range be rounding/saturated.

The conversion can be done by any of the convertors in SDP pipeline (except ew_inv_cvt).

CDP convertors

CDP has convertors listed below:

Convertor	Functionality	Limitation	Owner
cdp_in_cvt	Convert the input data compatible with LUT requirement, which means, the output of this convertor should be: $x*2^N$	For CDP.INPUT_DATA_TYPE=INT8/INT16 only	HW
cdp_lut_cvt	Each LUT entry has 16bits (can be interpreted as INT16 or FP16 based on pipeline), the original f(x) has to be converted to specified format to keep a high precision	No offset allowed	SW
cdp_out_cvt	Convert the results to INT8/16/FP16 before output to external	For CDP.INPUT_DATA_TYPE=INT8/INT16 only	HW

Suppose the CDP input has, in order to make LUT input has the form of $x*2^M$, cdp_in_cvt has to be programmed as:

$$O_{cdp\_in\_cvt} = -O_{in} * SF_{in}$$

$$SF_{cdp\_in\_cvt} = \frac{2^M}{SF_{in}}$$

Value M should be selected by precision study.

Suppose CDP output is encoded as O_{out}, SF_{out}, cdp_lut_cvt and cdp_out_cvt has to be programmed as:

$$O_{out} == \frac{O_{cdp\_out\_cvt}}{SF_{cdp\_lut\_cvt} * 2^M}$$

$$SF_{out} == SF_{cdp\_lut\_cvt} * SDP_{cdp\_out\_cvt} * 2^M$$

PDP convertors

There's no convertor instanced in PDP. But be noticed that the PDP padding value is intended to compensate the input offset, for FP16 pipe, they're ignored as we assume there's no offset for FP16 pipe;

Convertor statistics

NVDLA implemented counters to evaluate number of samples overflowed during convertor. The overflow is defined as:

$$INT32 : x < -2147483648 || x > 2147483647$$

$$INT16 : x < -32768 || x > 32767$$

$$INT8 : x < -128 || x > 127$$

$$FP16 : fabs(x) >= 65504$$

Here's a list of saturation counters in NVDLA pipeline:

Register	Valid condition
CACC. D_OUT_SATURATION	Always enabled
SDP.D_PERF_OUT_SATURATION	PERF_SAT_EN=YES && PROC_PRECISION== OUT_PRECISION==FP16
CDP.D_OUT_SATURATION	Always enabled