Gradient Code Generation for AI Accelerators with Specialized Data Layout Requirements

Authors: Hoai-Linh Tran, ZiChun Ye,
Giancarlo Colmenares,
Kai-Ting Amy Wang

Contents

Introduction

- This work presents an alternative approach of generating convolution backward kernels with respect to data and weights inputs using the forward convolution kernel which consists of image-to-column (im2col) and matrix multiplications:
  - Convolution is the most popular and extensively optimized operator in modern deep neural networks.
  - Current method of computing the convolution data gradient suffers from low efficiency with the usage of the column-to-image (col2im) function which performs multiple-to-one gradient aggregation.
  - Additionally Huawei's DaVinci processor has a hardware solution to further speed up the Im2Col operations.

Introduction

- The new approach requires non-trivial data format or layout conversions on the inputs and outputs. An iterator-based method will be presented to perform these required data conversions. We illustrate the iterator method using Huawei DaVinci's specialized tensor data layout.

Convolution Gradient Kernel Code Generation

A. Convolution gradient for tensors in NCHW format

$$Y = Conv2D(X, K)$$

Convolution parameters:

$$stride = s_f; dilation = d_f; padding = p_f$$

Backward “input”:

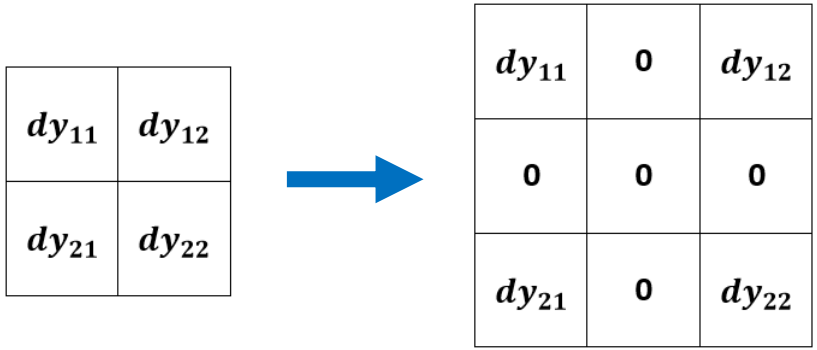
$$Head = \frac{dLoss}{dY}$$

A. Gradients for tensors in NCHW format

Gradient for convolution input (data channel):

$$dX = Conv2D\left(Strided\left(Head\right), FlipRot\left(K\right)\right)$$

Strided(Head) with (s_f -1 = 1)



FlipRot(K): Flip (transpose) dimensions N & C, rotate (180^o) in dimensions H & W

A. Gradients for tensors in NCHW format

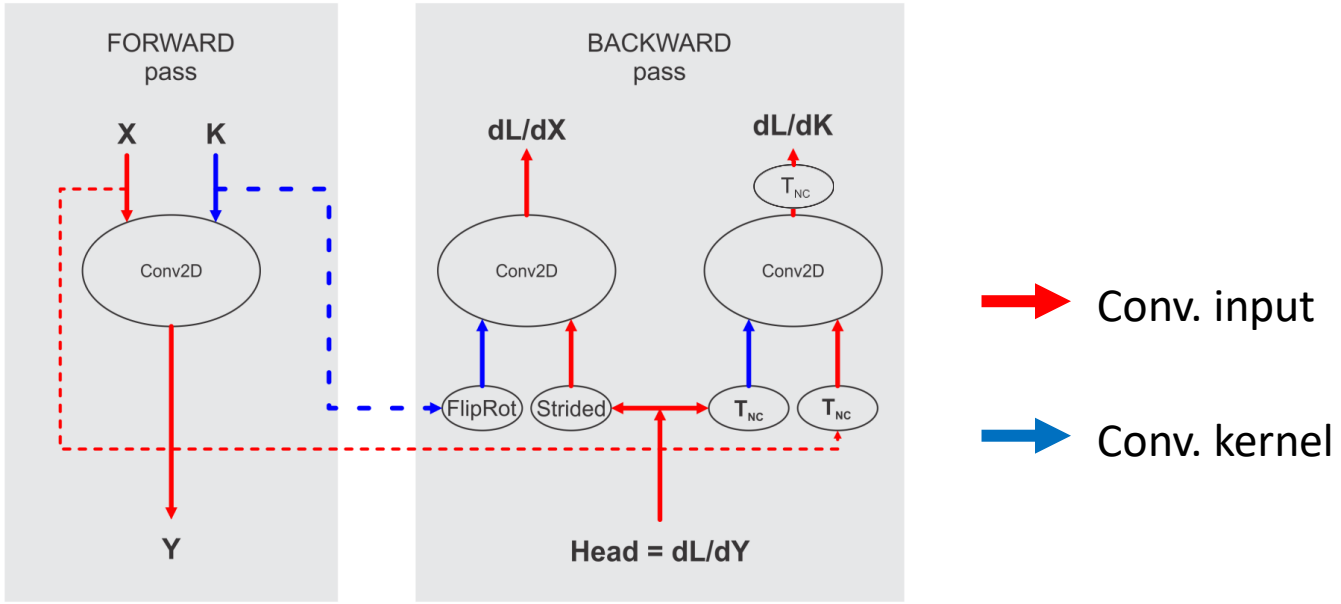
Gradient for convolution kernel (weight channel)

$$dK = T_{NC}\left(Conv2D\left(T_{NC}\left(X\right), T_{NC}\left(Head\right)\right)\right)$$

T_{NC}(X): Transpose dimensions N & C of tensor X

A. Gradients for tensors in NCHW format

Full scheme of computations for both gradients:



B. Gradients for tensors in fractalized format

Fractalized data formats for tensors in Huawei's DaVinci processors:

5D format for input:

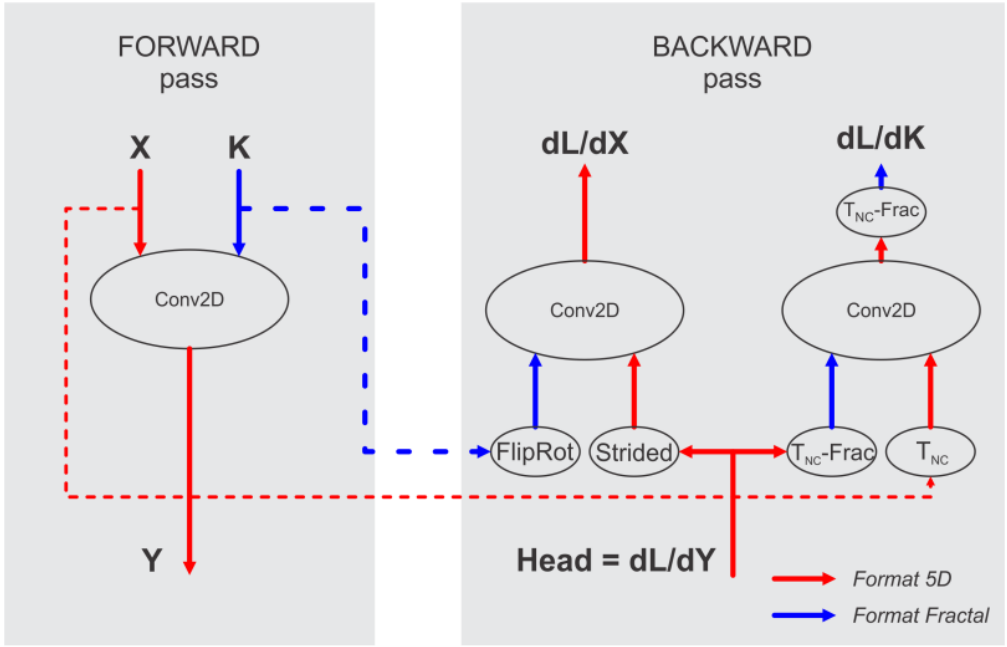
$$(N, C, H, W) \rightarrow (N, \frac{C}{16}, H, W, 16)$$

4D fractal format for kernel:

$$(N, C, H, W) \rightarrow (\frac{C}{16_c} \cdot H \cdot W, \frac{N}{16_n}, 16_n, 16_c)$$

B. Gradients for tensors in fractalized format

Full scheme of computations for both gradients:



B. Gradients for tensors in fractalized format

Iterator method to implement data converters

1. T_{NC} (transpose N and C dimensions of a 5D input, output is also in 5D)

$$X^{T_{NC}}[j_0, j_1, j_2, j_3, j_4] = X[i_0, i_1, i_2, i_3, i_4]$$

where

$$\begin{aligned} i_0 &= j_1 * 16 + j_4, \\ i_1 &= \lfloor j_0/16 \rfloor, \\ i_2 &= j_2, \\ i_3 &= j_3, \\ i_4 &= j_0 \% 16. \end{aligned}$$

B. Gradients for tensors in fractalized format

Iterator method to implement data converters

2. T_{NCf} (transpose N and C dimensions of a 5D input, output is in 4D fractal format)

$$X^{T_{NC_f}}[j_0, j_1, j_2, j_3] = X[i_0, i_1, i_2, i_3, i_4]$$

where

$$\begin{aligned} i_0 &= \lfloor j_0/(HW) \rfloor \cdot 16 + j_3, \\ i_1 &= j_1, \\ i_2 &= \lfloor j_0/W \rfloor - \lfloor j_0/(HW) \rfloor \cdot H, \\ i_3 &= j_0 \% W, \\ i_4 &= j_2. \end{aligned}$$

B. Gradients for tensors in fractalized format

Iterator method to implement data converters

3. FlipRot: (transpose N-C dimensions and rotating in H-W dimensions of a 4D fractal input, output is also in 4D fractal)

$$X^{FR}[j_0, j_1, j_2, j_3] = X[i_0, i_1, i_2, i_3]$$

where

$$\begin{aligned} i_0 &= j_1 \cdot HW + HW - 1 - j_0 \% (HW), \\ i_1 &= \lfloor j_0 / (HW) \rfloor, \\ i_2 &= j_3, \\ i_3 &= j_2. \end{aligned}$$

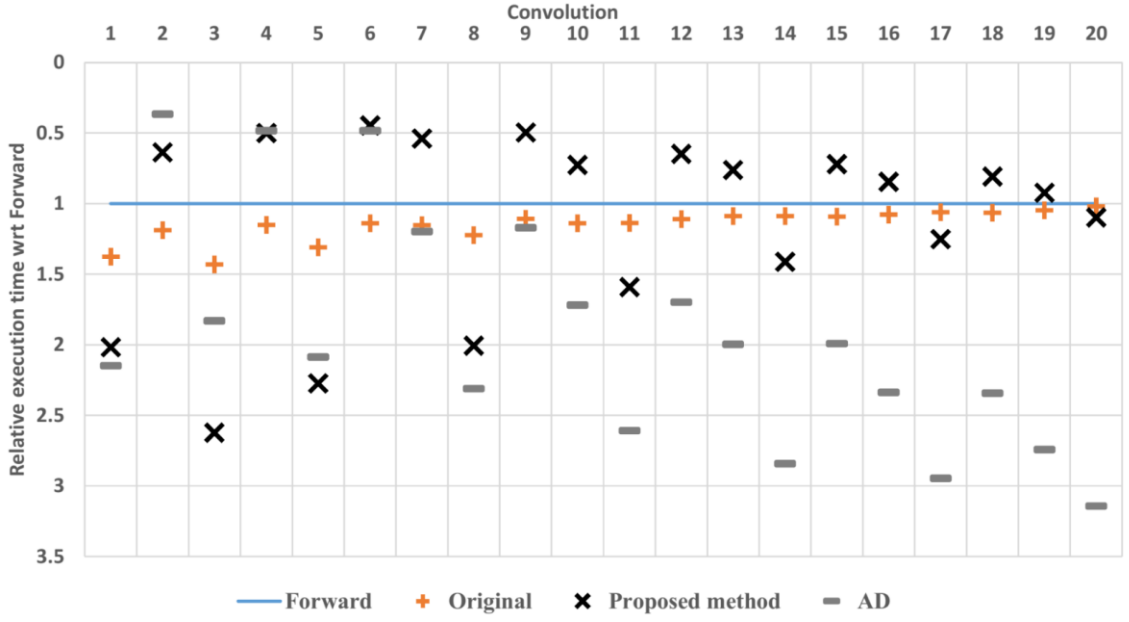
Numerical experiments and results

- Tested with 20 convolution shapes from ResNet-50
- CPU kernels
- Results compared with:
  - Backward kernels using Col2Im
  - Backward kernels auto-generated using TVM autodiff module

Layer	Input	weight
1	(1, 3, 224, 224)	(32, 3, 3, 3)
2	(1, 32, 112, 112)	(16, 32, 1, 1)
3	(1, 16, 112, 112)	(96, 16, 1, 1)
4	(1, 96, 56, 56)	(24, 96, 1, 1)
5	(1, 24, 56, 56)	(144, 24, 1, 1)
6	(1, 144, 56, 56)	(24, 144, 1, 1)
7	(1, 144, 28, 28)	(32, 144, 1, 1)
8	(1, 32, 28, 28)	(192, 32, 1, 1)
9	(1, 192, 28, 28)	(32, 192, 1, 1)
10	(1, 192, 14, 14)	(64, 192, 1, 1)
11	(1, 64, 14, 14)	(384, 64, 1, 1)
12	(1,384, 14, 14)	(64, 384, 1, 1)
13	(1,384, 14, 14)	(96, 384, 1, 1)
14	(1, 96, 14, 14)	(576, 96, 1, 1)
15	(1, 576, 14, 14)	(96, 576, 1, 1)
16	(1, 576, 7, 7)	(160, 576, 1, 1)
17	(1, 160, 7, 7)	(960, 160, 1, 1)
18	(1, 960, 7, 7)	(160, 960, 1, 1)
19	(1, 960, 7, 7)	(320, 960, 1, 1)
20	(1, 320, 7, 7)	(1280, 320, 1, 1)

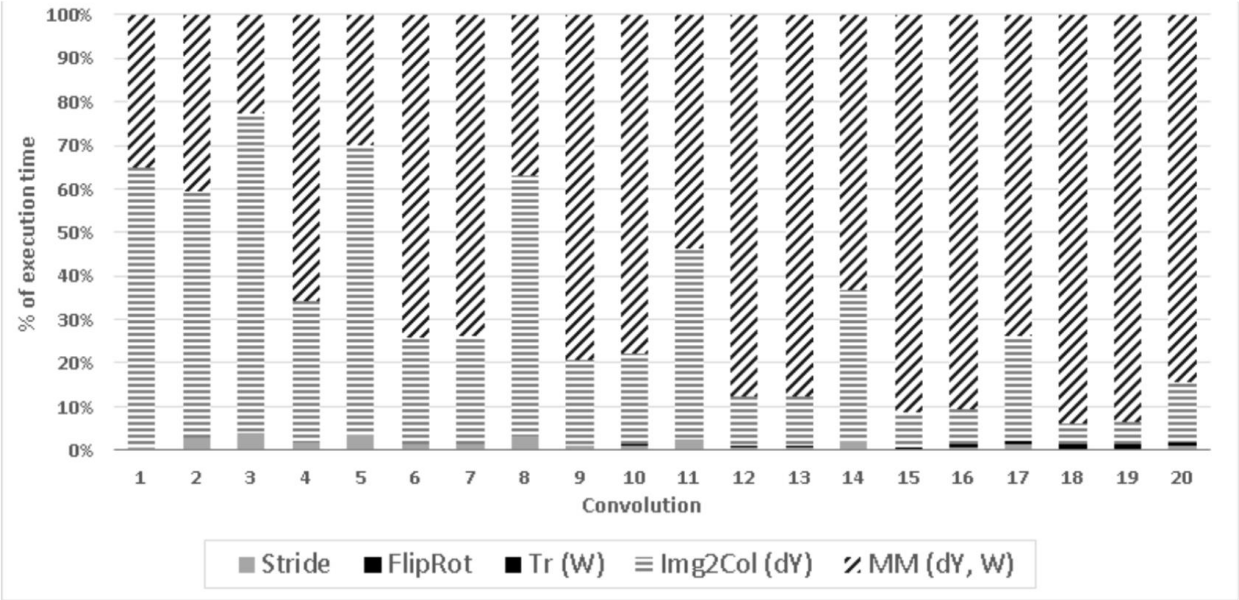
Numerical experiments and results

Comparison of the relative execution time of the considered backward convolution w.r.t. input approaches versus the forward convolution (above the line is faster)

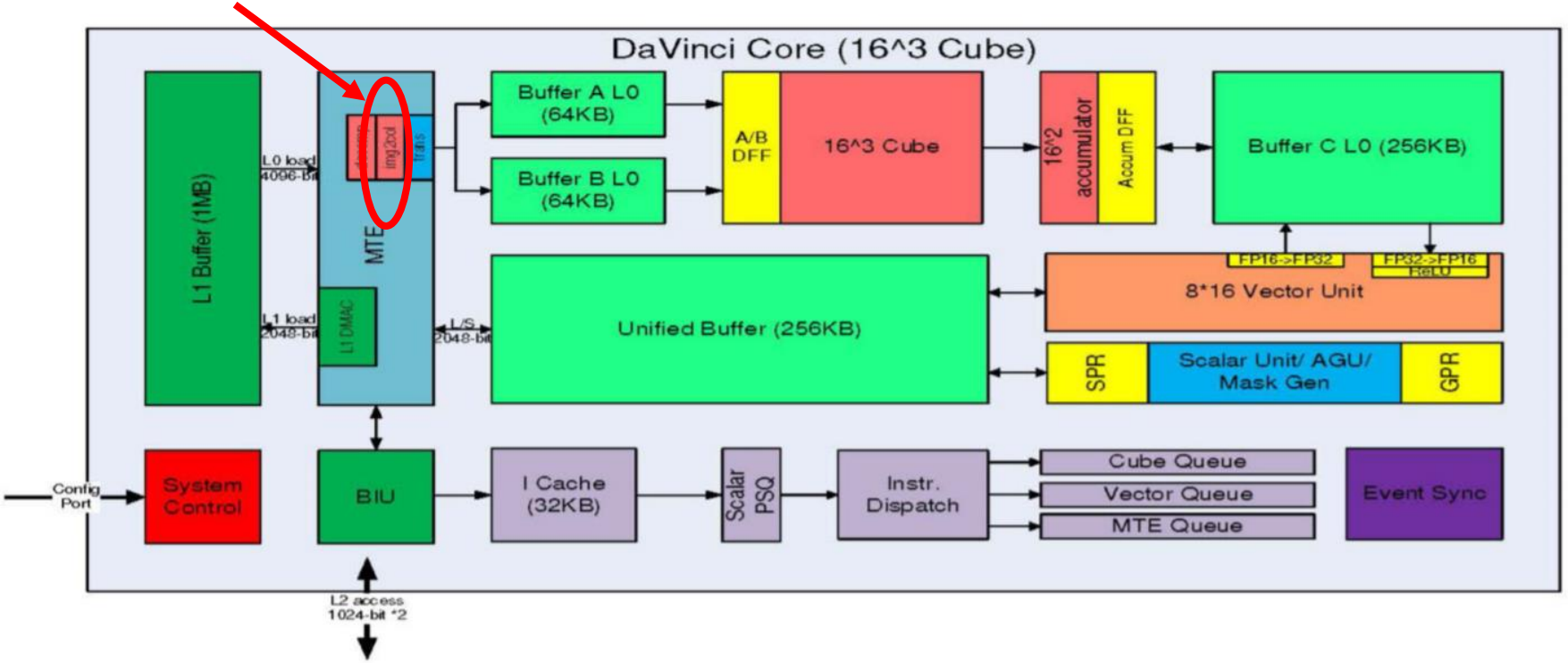


Numerical experiments and results

Percentage of execution time used by each operation of the proposed backward convolution function over the considered convolutions

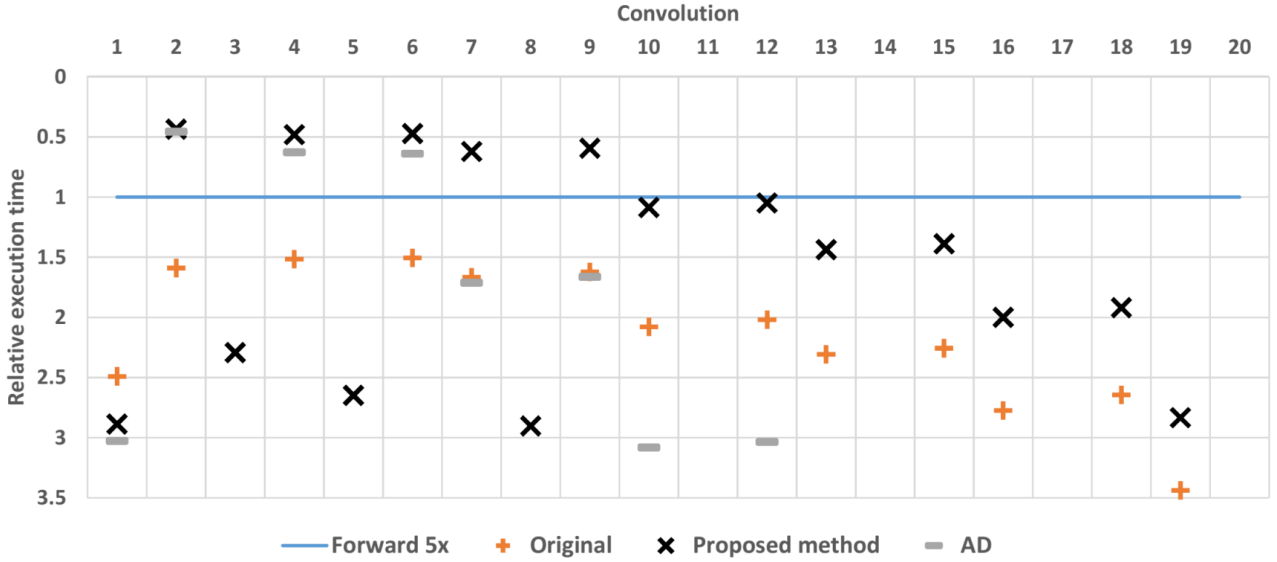


Numerical experiments and results



Numerical experiments and results

Comparison of relative execution time



Numerical experiments and results

- Without hardware support for Im2col, most (12/20) of backward kernel for input are faster than the version using Col2im.
- With hardware support for Im2col (approx. 5x acceleration), in 18/20 cases the kernel generated by the proposed method is faster.

Conclusions

- The backward convolution can be performed using forward convolution with data pre-processing via the data converters.
- The formulas used in the converters were delivered and tested to demonstrate correctness of the converters.
- The iterator method, to systematically express output iterators as functions of input iterators, were introduced to create converters for AI accelerators with non-trivial data format requirements,
- We showed that our approach is able to create efficient backward kernel with respect to input data.

Future works

- Gather performance for the gradients of K,
- Gather real performance numbers from Huawei's AI accelerators,
- Test the proposed method with convolution operators' shapes from other well known deep neural networks.

Thank you!

Gradient Code Generation for AI Accelerators with Specialized Data Layout Requirements

Authors: Hoai-Linh Tran, ZiChun Ye,
Giancarlo Colmenares,
Kai-Ting Amy Wang