Title: Characterizing and Demystifying the Implicit Convolution Algorithm on Commercial Matrix-Multiplication Accelerators

JX-Ma

2024/11/23

ABSTRACT

This article proposes an implicit im2col algorithm for Google TPU that is memory efficient and hardware friendly. It dynamically converts convolution into almost zero performance and memory overhead GEMM, fully unleashing the capabilities of the GEMM engine.

INTORDUCE

Commercial GPUs generally use implicit im2col algorithm to avoid performance and memory overhead in explicit algorithms. However, the exact implicit algorithm has not been made public. This article will reveal a hardware friendly and memory efficient implicit im2col algorithm used in TPU, which dynamically converts convolution into GEMM with almost zero performance and memory overhead, fully unleashing the functionality of the GEMM engine. This implicit algorithm utilizes the associative and commutative laws in convolution and requires optimization of memory layout and flattening strategies. Therefore, while providing data from simple single line memory, external memory access and computation are allowed to completely overlap.

IM2COL

The memory overhead from im2col is many times higher than the original matrix, as the transformation of im2col generates many reused elements.
This article uses the corresponding cuDNN API to measure the GPU execution time for both implicit and explicit versions. The execution time is normalized to the execution time of the implicit method. They found that explicit im2col is on average 28% slower than implicit methods. This is also consistent with the experimental results we conducted.

Channel-First Im2col Method

This article describes an optimization technique that involves placing an IFMap (Input Feature Map) in on-chip SRAM (Static Random Access Memory) to improve computational efficiency and avoid expensive multi bank SRAM and related crossbar switches. The idea of this layout is to send each IFMap element to a specific processing unit, which can accelerate the algorithm's execution process by pre computing and storing index information. Specifically, an example of laying IFMap in SRAM is described, where each row is an unfolded vector of a channel, called channel first or HWC layout. Through this layout, a 3x3 convolutional layer can be reduced to a GEMM (General Matrix Multiply) operation, and the process of reading columns from SRAM and how to ensure correctness are explained in detail.

In addition, it is recommended to use HWC format when storing IFMaps in DRAM (Dynamic Random Access Memory) to improve DRAM bandwidth utilization. We compared the access mode differences between IFMap storage layouts based on CHW and HWC, and pointed out that the HWC format can better utilize off chip storage bandwidth.

SUMMARY

The index pre calculation in this article first expands the input tensor and convolutional kernel tensor according to channels, and the data layout is NHWC layout. Then, the index of each element in the width of each input tensor is accessed. Compared with im2win, our data layout has better continuity and uses less space for index storage than their approach.