APNN-TC: Accelerating Arbitrary Precision Neural Network on Ampere GPU Tensor Cores

Boyuan Feng*, Yuke Wang*, Tong Geng, Ang Li, Yufei Ding.

* indicates equal contribution
Motivation

Quantized Neural networks

- Low cost (e.g., memory & computation)
- **Arbitrary precision** (e.g., int2, int3)

GPU Hardware

- Suitable for NN computation, especially with Tensor Cores
- Only support a **limited set of precisions** (e.g., int1, int4)
Key Ideas

• Support arbitrary precision neural networks with the limited precisions on Tensor Cores
  • Utilize bit-level operations (e.g., XOR and AND)
Challenges

• Lack of mathematical emulation design
  • For supporting multiplication and addition in quantized NNs with only bit-level operations
  • For supporting diverse input data (e.g., -1/+1 or 0/1)

• Lack of efficient implementation for arbitrary precision NN layers
  • Need to exploit data locality in our emulation workload
  • Need specialized bit operations and data organization to avoid uncoalesced memory access

• Lack of efficient NN framework designs
  • To exploit data reuse opportunity across NN layers (e.g., location of quantization layer)
Overview of APNN-TC
AP-Bit Emulation Design

- AP-Bit Operation Template Design

Cost Analysis (p-bit W and q-bit X of shape n×n)

- Bit decomposition: $O((p+q)n^2)$ [Negligible]
- TC computation: $O(pqn^3)$
- Bit combination: $O(pqn^2)$ [Negligible]
AP-Bit Emulation Design

• Data Adaptive Operator Selection

Problem:
• Bit-0 and bit-1 may encode diverse values
• 1-bit weight matrix may encode -1 and 1, instead of 0 and 1
  • I.e., bit-0 -> -1, bit-1 -> 1
• Naïvely utilizing AND bit operation leads to erroneous computation results

Our design:
• Adaptively select operator based on data encoding information
• Case-I:
  • When both W and X encode 0 and 1
  • Use logical AND operation
  • Example:
    • W = [0, 1], X = [1, 1]
    • WX = \text{popc}(\text{AND}([0, 1], [1, 1]))
    • = \text{popc}([0, 1]) = 1
• Case-II:
  • When both W and X encode -1 and 1
  • Use logical XOR operation
  • Example:
    • W = [-1, 1], X = [1, 1]
    • WX = n - 2*\text{popc}(\text{XOR}([0, 1], [1, 1]))
    • = n - 2*\text{popc}([0, 1]) = 1
• Case-III:
  • When W encodes -1 and +1, X encodes 0 and 1
  1) first transform W into a vector with 0/1
  2) Compute with logical AND operation
  3) Recover the value WX with linear transformation
  • Example:
    • W = [-1,1], X = [1,0]
    • W’ = (W+[1,1])/2 = [0,1]
    • WX = 2W’X - [1,1]X = -1
Arbitrary Precision Layer Design

- Arbitrary-Precision Matrix Multiplication (APMM)
Arbitrary Precision Layer Design

- Arbitrary-Precision Convolution (APConv)
Arbitrary Precision Layer Design

• Input-aware Padding Design

Problem:
• Bit-0 and bit-1 may encode diverse values
  • E.g., weight W encodes -1 and 1 with 0 and 1
  • Cannot naively padding 0 since 0 represents -1

Solution:
• Case-I: both weight and feature encode 0/1
  • Simply pad 0 for features
• Case-II: both weight and feature encode -1/1
  • Pad 1 for features
  • Use an extra counter to track the number of weight 0’s outside the input image frame
• Case-III: weight encodes -1/1 and feature encodes 0/1
  • Pad 0 for features
Arbitrary Precision Layer Design

• Performance Analysis

Performance Model:
• Consider tread-level parallelism (TLP) and compute intensity (CI)
• Given:
  • a p-bit weight matrix of shape M × K
  • a q-bit feature matrix of shape K × N
  • matrix tiling size b_m × b_n
• We have:

\[
TLP = \frac{pM \times qN}{b_m \times b_n}
\]

\[
CI = \frac{2 \times b_m \times b_n}{b_m + b_n}
\]

Auto-tuning:
• A heuristic algorithm to find the design with largest CI while maintaining TLP
Arbitrary Precision Neural Network Design

Key insights:

• Given 32-bit output from Tensor Cores, we quantize the TC computation results before writing to global memory

• Fuse APMM/APConv with following quantization, BN, pooling and ReLU kernels to minimize global memory access

• For scalar operations (e.g., ReLU), reduce shared memory access by directly reusing values in registers
Evaluation

• APMM Performance
Evaluation: APNN Inference

- APNN Inference Performance

<table>
<thead>
<tr>
<th>Schemes</th>
<th>ImageNet-AlexNet</th>
<th>ImageNet-VGG</th>
<th>ImageNet-ResNet18</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8 Latency</td>
<td>Throughput</td>
<td>8 Latency</td>
</tr>
<tr>
<td>CUTLASS-Single</td>
<td>25.22ms</td>
<td>3.29×10^2fps</td>
<td>116.84ms</td>
</tr>
<tr>
<td>CUTLASS-Half-TC</td>
<td>14.37ms</td>
<td>6.21×10^2fps</td>
<td>31.42ms</td>
</tr>
<tr>
<td>CUTLASS-INT8-TC</td>
<td>3.78ms</td>
<td>2.40×10^3fps</td>
<td>23.53ms</td>
</tr>
<tr>
<td>BNN</td>
<td>0.69ms</td>
<td>1.37×10^4fps</td>
<td>2.17ms</td>
</tr>
<tr>
<td>APNN-w1a2</td>
<td>2.87ms</td>
<td>3.79×10^3fps</td>
<td>7.50ms</td>
</tr>
</tbody>
</table>
Evaluation

- Overhead from bit combination and bit decomposition
Evaluation

• Speedup from Kernel Fusion
Questions?

The project is open-sourced at:

https://github.com/BoyuanFeng/APNN-TC