基于 CUDA/Triton 的 GPU 算子设计

Preliminary

Quantization

对于一个神经网络，量化可以用来减小模型的存储开销和计算开销，加快训练或者推理的速度。具体来说，量化是指用更少的比特精度（例如 INT8，INT4）来表示高精度(FP32/FP16)的权重和激活，可以用公式 1 来描述。

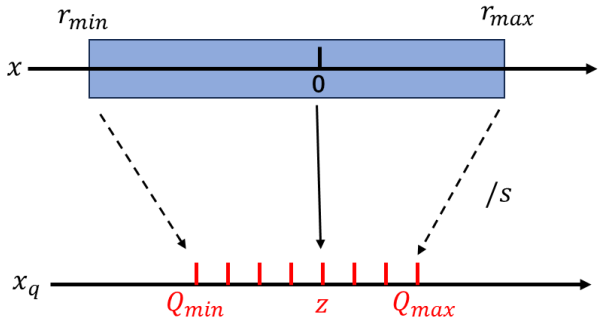

图 1 量化示意图

$$x_q = clamp\left(round\left(\frac{x}{s}\right) + z; Q_{min}, Q_{max}\right) \tag{1}$$

其中 clamp 函数的定义如下：

$$clamp(x; a, b) = \begin{cases} a, & if\ x < a \\ x, & if\ a \le x \le b \\ b, & if\ x > b \end{cases} \tag{2}$$

在公式 1 中，x是高精度的权重或者激活，x_q是量化后的整数。公式 1 中的s表示缩放因子，负责将$[r_{min}, r_{max}]$映射到$[Q_{min}, Q_{max}]$。 z表示零点，代表x中 0 的量化后的位置。如果x关于零点大概呈对称分布，我们使用对称量化，此时$z = 0$，公式 1 可以简化为

$$x_q = clamp\left(round\left(\frac{x}{s}\right); Q_{min}, Q_{max}\right) \tag{3}$$

s可以根据公式 4 进行计算

$$s = \frac{r_{max} - r_{min}}{Q_{max} - Q_{min}} \approx \frac{|r|_{max}}{Q_{max}} \tag{4}$$

其中Q_{max}, Q_{min}分别是量化后的最大值和最小值。给定量化位宽N，$Q_{min} = -2^{N-1}, Q_{max} = 2^{N-1} - 1$.

本实验主要对神经网络线性层进行量化，它是目前视觉和语言模型中最常用的操作。不考虑偏置，线性层可以用以下公式表示:

$$Y = XW^T \tag{5}$$

其中X表示激活，W表示权重。在量化时，我们可以根据整个$X(W)$确定一个缩放因子，称为 per-tensor 量化，但这种粗粒度的量化方式会带来较大的量化误差。所以我们常用更细粒度的量化方式，即对X的每一行或者W的每一列计算缩放因子，如图 2 所示。我们把这种量化方式称为对激活的 per-token 量化以及对权重的 per-channel 量化。本实验中采样的就是这种量化方式。

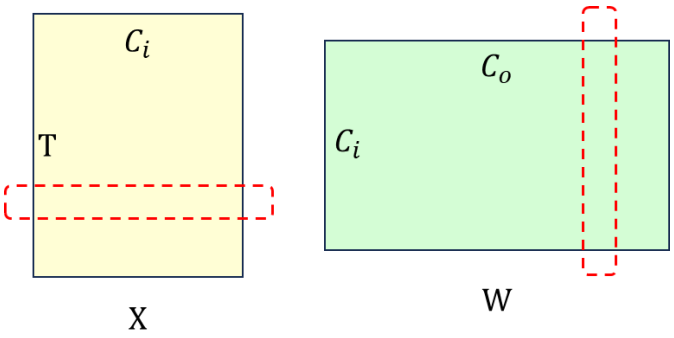

图 2 激活的 per-token 量化以及权重的 per-channel 量化

Sparsity

稀疏矩阵（Sparse Matrix）是指大部分元素为零的矩阵。为了节省内存和提高计算效率，稀疏矩阵通常不直接存储其所有元素，而是采用某种特定的存储格式来仅存储非零元素。常用的稀疏矩阵的表示形式有 Coordinate list (COO), Compressed Sparse Row (CSR), Compressed Sparse Column (CSC), Block sparse row (BSR). 本实验中的稀疏矩阵采用 BSR 格式，是以 block 为单位存储非零元素，具体定义可以参考下列链接：https://quansight.com/post/pytorch-2-

1-quansights-improvements-to-bsr-sparse-matrix-multiplication/
在进行稀疏矩阵乘法运算时，可以跳过为 0 的值或为 0 的 block 以达到减少计算的目的。尽管 CUDA 提供的 cuSPARSE 支持稀疏矩阵乘法，但只支持 fp32，不支持 fp16 格式的输入。因此本实验需要同学们实现一个支持 fp16 输入的 sparse MM kernel.

Triton

Triton 是一个由 OpenAI 开发的高效深度学习编程语言和库，旨在简化 GPU 加速的自定义运算核的编写。Triton 使得研究人员和工程师能够在不需要深入了解 CUDA 编程的情况下，快速构建高效的 GPU kernel，优化深度学习模型的训练和推理。它采用类似 Python 的语法，但同时能够生成高度优化的 GPU 代码。通过自动调优和并行计算，Triton 使得用户能够显著提高计算效率。本次实验需要同学们完成量化矩阵乘 kernel (W8A8, W4A16)以及稀疏矩阵乘 kernel.

安装的 triton 版本为 2.1.0，同学们可以根据 triton 官方的 tutorial 进一步学习，主要理解其中 matrix multiplication 的部分，后续的 kernel 实现都是在此基础上的修改。
https://triton-lang.org/main/getting-started/tutorials/index.html

实验要求

W8A8 MM

我们已经定义好了 QuantizedLinear_w8a8 这个类，在这个类中已经预先完成了对权重的量化，得到 INT8 表示的 qweight，以及权重对应的缩放因子 weight_scale。同时，在 forward 函数中我们也完成了对于激活缩放因子的计算，得到了 input_scale。
需要同学们完成 W8A8_MM kernel，它用到的输入是：FP16 表示的激活, qweight 的转置，input_scale，weight_scale.
提示：在 kernel 应该完成三个操作: 1.对 FP16 activation 进行量化；2.对量化后的 activation 和 weight 进行整数矩阵乘法； 3. dequantize，也就是把 2 得到的结果乘以 activation 和 weight 各自的 scale 得到最终的结果。

W4A16 MM

我们已经定义好了 QuantLinear_w4a16 这个类，在这个类中已经预先把权重量化到 INT4，并且将同一列中的 8 个 INT4 的权重打包到一个 INT32 表示的数中，得到 qweight。我们还

得到了权重对应的缩放因子 scales 的转置，它是一个长度等于权重矩阵W列数的列向量。需要同学们完成 W4A16_MM kernel，它用到的输入是：FP16 表示的激活, qweight，scales. 提示：在这个 kernel 中，首先需要把 weight 从 INT32 表示的 qweight unpack 出来，再 dequantize（即乘以对应的缩放因子），然后与 FP 16 的 activation 相乘。 注意 unpack 的时候不能直接用 (weight >> 4) & 0xF，因为这样得到的结果是无符号数（范围是 0~15）， 而我们需要的是有符号的 int4 (范围是-8~7)

Sparse MM

本实验需要是实现的是 dense = dense x sparse (DDS)矩阵乘， 其中 Sparse matrix 是 BSR 格式表示的。我们已经预先定义好了 SparseLinear 这个类，需要同学们完成 DDS_MM kernel，它用到的输入是 fp16 dense activation 以及 BSR 格式表示的 weight. 这里我们规定 weight 的维度是[1024, 1024] 需要同学们实现 BSR 格式中 block size 为 32 的 DDS_MM kernel