第19次课程-llama模型中的其余算子实现

在前面的几次课程中，我们分别用Cuda高效的实现了RMSNorm、GEMV、kv-cache

自制大模型推理框架

带你从零写一个支持LLama2/3推理
支持Cuda加速和int8量化的大模型框架

课程亮点

一、项目整体架构和设计

学习架构思维，我们将在这里学习算子、张量等关键数据结构的设计。

二、支持LLama2模型结构

对LLama模型的权重读取和构建，用KV Cache等机制支持加速推理。

三、模型的量化

为了减少显存的占用，我们会一起开发int8模型量化模块，包括量化模型导出和量化算子支持。

四、Cuda基础和算子实现

带你学Cuda并实战大模型算子的实现，每个算子都将讲解原理，从零手写。

五、用推理框架做点有趣的事情

文本生成，生成故事，多轮对话。

现价优惠　扫码马上报名

视频

llama模型中的其余算子实现.mp4

“Hello，my”

32表示hello

[512维度]向量

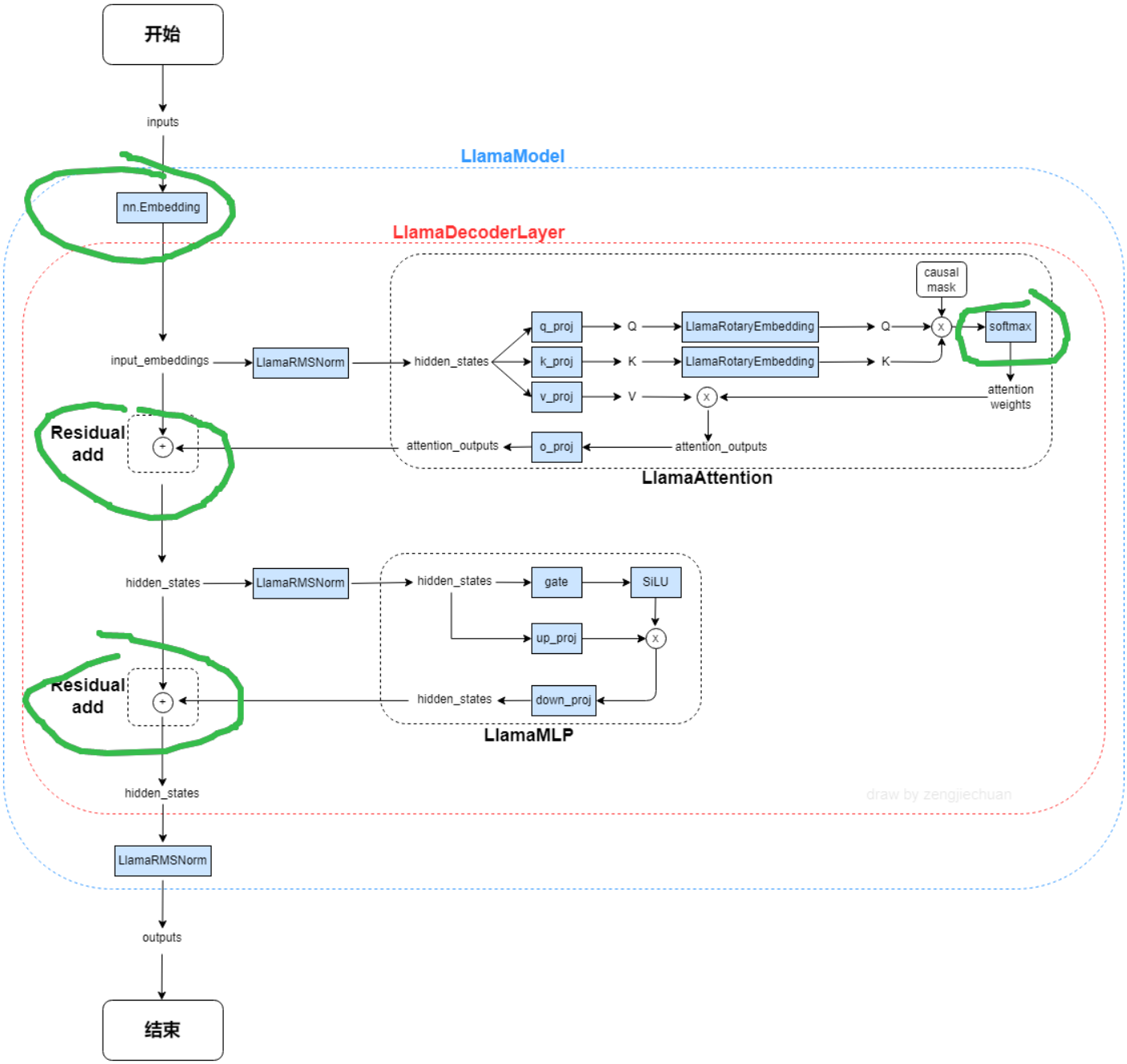
99表示，

[512维度] 向量

37表示my

Inputs = [32,99,37]

[3, 512]，维度为3×512输入向量



学习本课程前请先参考上一节课程：

第18次课程-用Cuda实现LLama模型中多头注意力算子

Add算子的实现

在学习的过程中，add算子可能是我们接触到的一个最简单算子，它的作用就是将两个输入的张量进行逐个元素的加和并将结果写入到输出张量的对应位置中。

```
1   __global__ void add_kernel_cu_fp32(int32_t size, const float* in1, const
    float* in2, float* out) {
2     int32_t tid = threadIdx.x + blockDim.x * blockIdx.x;
3     if (tid >= size) {
```

```
    return;
  }
  float in_val1 = in1[tid];
  float in_val2 = in2[tid];
  out[tid] = in_val1 + in_val2;
}
```

现在分别有两个指向输入的in1和in2指针，其中数据的数量为size个，我们配置了n个线程对size个数据进行逐个相加的工作。假设size为1024，线程数量n将它配置等于1024，总共有32个线程块，这样一来每个线程块中的线程数量是32。这里因为数据量不是特别多，所以我们配置一个线程处理一个位置元素的相加计算

```
int32_t tid = threadIdx.x + blockDim.x * blockIdx.x;
```

blockIdx.x表示线程块的编号，在上例中就是从0-31，因为总共有32个线程块，而线程块的容量为32（表示每块有4个线程），也就是blockDim.x等于32. 我们用python来模拟这个过程，可以看出线程号计算依次打印出0-1024。

```
In [2]: blockDimx = 32 线程块的维度

In [3]: blockCount = 32 线程块的个数

In [4]: for blockIdx in range(blockCount):
   ...:     for threadIdx in range(blockDimx):
   ...:         print(threadIdx + blockDimx * blockIdx)
   ...:
0
1
2
3
4
5
6
7
8
...
...
...
1023
```

所以`int32_t tid = threadIdx.x + blockDim.x * blockIdx.x`的值依次是0到1023，每个线程负责处理一个位置的输入数据。

```
float in_val1 = in1[tid];
float in_val2 = in2[tid];
out[tid] = in_val1 + in_val2;
```

随后这多个线程分别取出对应位置tid的值，in_val1和in_val2进行相加并将值放入到输出out[tid]中，另外我们要知道和CPU计算不同的是，这1024个线程不是依次执行的，而是并行执行的，也就是线程id从0到1023并没有执行的先后顺序。

Embedding算子的实现

比如现在有一个大小为65200×512的词表，65200就表示对应输入的单词数量，512就表示映射后的向量维度，我们以下图作为一个例子，图中表的一行表示一个单词，表中的一列表示单词对应的向量。

1	2	3	4
4	3	1	7
6	1	3	4

Hello,Hi --> 0,3,6 id array

以以上的表格为例我们的单词数量为8，每个单词的维度是4。如果我们的输入编号是[0，3，6]，那么我们Embedding算子将取出对应位置的单词向量，以上表格为例，取出对应的向量为：

[1,2,3,4]

[4,3,1,7]

[6,1,3,4]

综上来说，我们需要为这个算子准备三个输入，一个是词表权重，在本例中就是这个以上的8×4大小的表格，以及需要取出的输入编号，也就是[0,3,6]，最后一个输入将用于索引到的权重向量，也就是

以上的三个向量。**所以在Cuda算子的实现中，我们配置了N个线程块，N表示单词的数量，每个线程块中的线程数量为M，M个线程会负责搬运每个单词对应的weight_dim个数值。**

```
__global__ void emb_kernel_cu_fp32(int32_t vocab_size, int32_t token_num, int32_t weight_dim,
                                   const int32_t* input_ptr, const float* weight_ptr,
                                   float* output_ptr) {
  int32_t token_idx = blockIdx.x; // 0,1,2
  if (token_idx >= token_num) {
    return;
  }
  int32_t token = input_ptr[token_idx]; // 0,3,6
  if (token >= vocab_size) {
    return;
  }
    //  输出位置 = 起始位置 + 1*4
  float* output_ptr_start = output_ptr + token_idx * weight_dim;
  // 起始位置+3*weight_dim(4)
  const float* weight_ptr_start = weight_ptr + token * weight_dim;
    // 定位到词表的一行之后，我们开始搬运。
  for (int32_t i = threadIdx.x; i < weight_dim; i += blockDim.x) {
    output_ptr_start[i] = weight_ptr_start[i];
  }
```

换句话说，我们需要将input token size×weight dim个数据复制或者说搬运到输出空间中，那么我们就配置input token size个线程块，每个线程块中的所有线程会一起搬运对应的weight dim个权重数据。

```
  int32_t token_idx = blockIdx.x;
  if (token_idx >= token_num) {
    return;
  }
```

获取到当前线程块需要处理的单词编号，对于上例的输入来说，token_idx的值依次是0，3，6。从以下的代码的代码中定位到所需要取的词表行：

```
  float* output_ptr_start = output_ptr + token_idx * weight_dim;
  const float* weight_ptr_start = weight_ptr + token * weight_dim;
```

weight_ptr + token * weight_dim; 这行代码表示我们现在需要定位到词表的某一行，同样地，output_ptr_start指向输出张量某一行的起始地址，比如现在我们要搬运词表中的第0行，[1,2,3,4] 那么weight_ptr_start就指向1所在的位置。

```
    for (int32_t i = threadIdx.x; i < weight_dim; i += blockDim.x) {
      output_ptr_start[i] = weight_ptr_start[i];
    }
```

随后就是将每个单词对应的dim维度的向量搬运到对应的输出张量中。

softmax算子的实现

这里的softmax算子用于自注意力机制中用于对一个头中的score值求激活值，换句话说，如果有多个自注意力头，那么我们将调用softmax对每个头求激活值。就像上文说的那样：

query的维度为(bsz, 1, heads, head_dim)，key的维度为(bsz,1+cache_len, heads, head_dim)，此处bsz恒定等于1。所以query的维度是(1, 1, heads, head_dim)，对于某个头来说，query一个头(head)的维度就是**(head_dim)**。

而自注意力计算中的key矩阵维度分别为批次大小，历史步长cache_len+当前步长1，heads（多头的头数）以及每个头的维度，对于key矩阵的当前头key head，key head的维度是**(1+cache_len,head_dim)**，随后根据自注意力公式的要求，随后要做的事情就是将query head 矩阵乘 key head，得到当前头的scores矩阵，也就是：

q_head matmul **key_head = scores head**

(1,head_dim)@(head_dim, 1+cache_len) = score head = (1,cache_len+1)

score_head 维度是(1,cache_len+1)，随后我们要做的事情就是将这里得到的score_head求softmax，另外有一点需要提醒，在我们的cuda核函数中，每个线程块block处理一个head，所以我们每个block都会有一个维度为(1,cache_len+1)的score_head，并需要对它求取softmax.

```
__device__ void softmax_gpu(float* __restrict__ x, int size) {
  int tid = threadIdx.x; // block 0,有线程0, 1, 2, 4, ...31
  int step = blockDim.x; // step = 32

  // find max value (for numerical stability)
  float max_val = tid < size ? x[tid] : 0;
  // tid = 0 , step = 32 = 线程块的维度
  for (int i = tid + step; i < size; i += step) {
    if (x[i] > max_val) {
      max_val = x[i];
    }
  }
  // x 对应就是score_head指针
```

```
}
```

该方法的x是score_head指针，size是cache_len+1。上面我们说到每个线程块(block)都要调用softmax,所以这里的threadIdx是从0到BlockDim，比如BlockDim等于32，那么这里的tid = threadIdx序号从0到31，step等于32.

此处的max_val是每个线程分别计算局部和，例如**线程0计算位置0，32，64, ... 位置**的局部最大值，**线程1计算位置1，33，65，。。。，等位置的局部最大值**，线程2计算位置2，34，66，96，。。。等位置的局部最大值，线程31计算位置，31，63，95等位置的局部最大值。

随后我们再将32个线程的局部最大值进行规约（这里的操作叫BlockReduceMax，在之前的课程中已经讲过了）

第5次课程-RMSNorm算子的CUDA实现

第6次课程-Nsight compute的使用和算子的优化

```
using BlockReduce = cub::BlockReduce<float, 128>;
__shared__ BlockReduce::TempStorage temp;
__shared__ float shared_val;
max_val = BlockReduce(temp).Reduce(max_val, cub::Max());
if (threadIdx.x == 0) {
  shared_val = max_val;
}
__syncthreads();
max_val = shared_val;// score head中的最大值
```

我们将属于32个线程的一个线程块内的局部和进行规约，得到一个block中的最大值对应的也就是一个score head中的最大值，一个score head的对应维度为**(1,cache_len+1)**，**这里的max_val就是这组数据中的最大值**。

我们用同样的方法先求得每个线程对应一组下标的局部和，再求出score head向量中输入值减去最大值再求e次方的全局和sum。

```
float sum = 0.0f;
for (int i = tid; i < size; i += step) {
  x[i] = expf(x[i] - max_val);
  sum += x[i];
}
sum = BlockReduce(temp).Sum(sum);
if (threadIdx.x == 0) {
  shared_val = sum;
}
__syncthreads();
```

```
11   sum = shared_val; // 一个score head中的最大值
```

最后再将原始输入除以全局和sum就能得到最终的结果，也就是一个score head经过softmax计算后的输出。

```
1   for (int i = tid; i < size; i += step) {
2     x[i] /= sum;
3   }
```