softmax

输入矩阵大小：`[M, K]`

Warp Reduction - When K is small

当K很小的时候，我们选择将K个元素使用一个warp内完成。

我们使用`__shfl_xor_sync`来同时完成一个warp内的寄存器元素的reduction。`xor`意味着使用`xor`来shuffle输入下标。

```
template <typename T>
__inline__ __device__ void warpReduceMax(
    T &val,
    int thread_group_width = WARP_SIZE)
{
    for (int mask = thread_group_width / 2; mask > 0; mask >>= 1)
    {
        val = fast_max(val, __shfl_xor_sync(FINAL_MASK, val, mask,
WARP_SIZE));
    }
}
```

上面代码中，`thread_group_width`是一组thread的数目，该组thread负责一行的max求值。`FINAL_MASK=0ffffffff, WARP_SIZE=32`。上面的操作中，利用`mask`可以刚好使得warp中的线程按照`thread_group_width`为长度切分的各个部分都分别进行reduction。例如`thread_group_width=16`：

```
第一次迭代：
mask=8: 1 0 0 0

0-7：由于第五位上为0，因此异或的结果为+8，所以0-7线程分别对应8-15线程
8-15：由于第五位上的值为1，所以异或的结果为-8，所以8-15对应0-7线程（这个没有意义，只要
0-7对应上就可以了）
16-23：同样由于第五位为0，所以+8，对应24-31

第二次迭代：
mask=4：1 0 0
0-3：由于第三位上为0，所以+4，对应4-7
16-19：由于第三位上为0，所以+4，对应20-23

第三次迭代：
mask = 1 0
0-1：由于第二位上为0，所以+2，对应2-3
16-17：由于第二位上为0，所以+2，对应18-19

第四次迭代：
mask = 1
```

```
0：由于第一位上为0，所以+1，对应1
16：由于第一位上为0，所以+1，对应17
```

综上，最终每个`thread_group_width`个线程中的第一个线程(0 and 16)都会得到最终的reduction结果，也就是每行的最大值。

我们使用向量读取，读取长度假设为`pack_size`。

- 当`K/pack_size >= 32`，我们启动M/pack_size blocks and 128 threads。每个block进一步被划分为x和y两个维度，沿着x维度我们执行warp内列方向的reduction，在y维度我们并行着独立的行操作。例如，我们启动`gridDim = M / pack_size, blockDim = <32, 4>`，每个线程处理`K/32`列元素，每个block处理4行，32列，每个grid处理`M/4`行。
- 当`K/pack_size < 32`，我们启动`M*K/pack_size/128` blocks和128 threads。每个block进一步被分为x和y两个维度，在x维度上我们执行warp reduction，y维度上我们并行着独立的行操作。但是这个情况下，我们执行warp reduction的warp size是`K/pack_size`。例如，我们启动`gridDim = M*K/pack_size/128, blockDim = <K / pack_size, 128 / K * pack_size>`，每个线程处理`pack_size`列元素，每个block处理`128 / K * pack_size`行，`K/pack_size`列，每个grid处理`M*K/pack_size`行。

Block Reduction - When K is large

理想情况下，warp内部的reduction是最快的。但是，我们需要寄存器来存储输入，每个线程需要存储`K/ 32 * pack_size * sizeof(dataType)`。**线程的寄存器资源有限**。当寄存器不足的时候，CUDA将自动使用shared memory，这使得warp reduction变得不太理想。当然，当`K`很大的时候，使用shared memory已经足够快了。

how to do block reduction?因为我们需要在整个block中完成reduction，所以我们需要方法在线程之间完成同步，例如，我们需要使用shared memory。我们在每个warp内完成warp reduction，然后存储到shared memory，接着load到第一个warp中，再次执行reduction。

例如，假设我们的`blocksize`为112，这可以分解为`32+32+32+16`。我们首先在每个warp内reduce，然后我们存储到shared memory中的warp_max数组中，with`index = threadIdx.x / warp_size`，当`threadIdx.x < (blockDim.x / 32)`，我们读取shared memory中的值到第一个warp内的寄存器，例如`threadIdx.x = 0,1,2,3`(因为112被分成了4个warp)，最后在第一个warp内完成reduction。

我们启动`M`blocks和1024（maximum）个线程，每个block处理一行。例如，我们启动`gridDim = <M>, blockDim = <block_size>, Shared memory = K * sizeof(dataType)`。 对于`block_size`的选择，可以是`1024, 512, 256, 128`。我们首先使用 `cudaOccupancyMaxActiveBlocksPerMultiprocessor` 来选择合适的block size，这个函数会根据提供的内核函数、块大小和共享内存需求来计算每个多处理器上可以同时运行的最大活动块数，最大活动块数是指一个多处理器（SM）上可以同时运行的最大线程块数。

```
cudaError_t err = cudaOccupancyMaxActiveBlocksPerMultiprocessor(
    &max_active_blocks_conf_1,
    softmax_block_smem<T, ACT_T, block_size_conf_1>,
    block_size_conf_1,
    smem);
```

选择`block_size`的过程：我们首先选择`128`作为block size，然后我们使用`cudaOccupancyMaxActiveBlocksPerMultiprocessor`来计算每个多处理器上可以同时运行的最大活动块数。接着，我们按照从`1024`到`256`递减的顺序，启动`cudaOccupancyMaxActiveBlocksPerMultiprocessor`，获取到最大活动块数。**由于后面三中的`block_size`大于`128`，所以最大活动块数肯定小于等于`128`，当我们从`1024`到`256`过程中，如果有一个`block_size`的最大活动块数等于`128`，那么我们就选择这个`block_size`。**

resource

1. AI Templete Github
2. How to write a fast Softmax CUDA kernel?
3. Oneflow 知乎