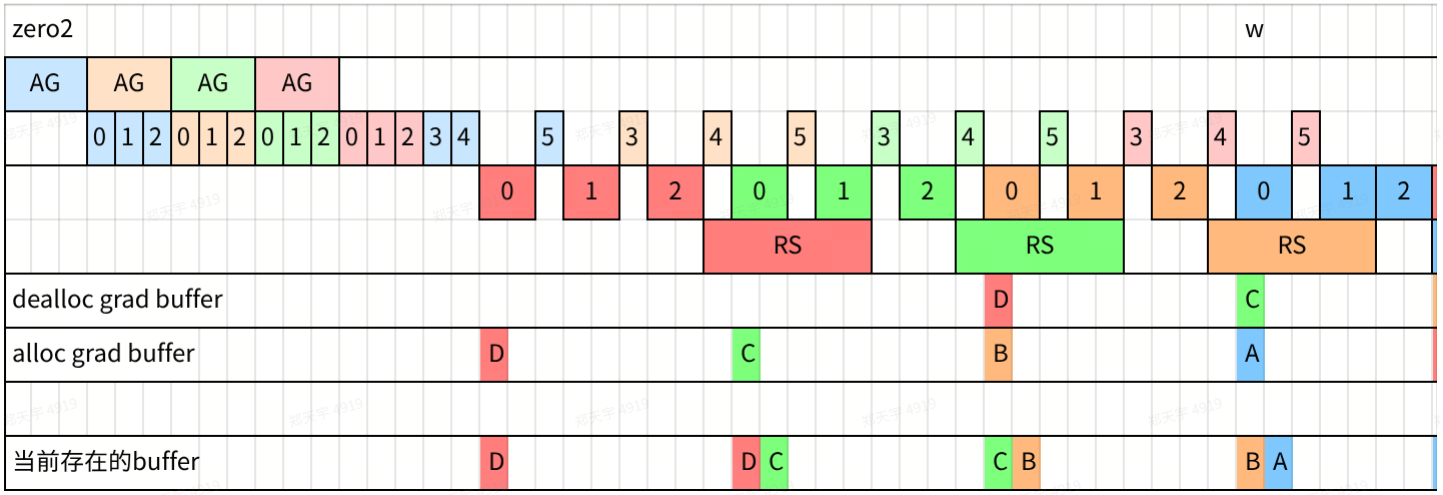
zero2_dealoc_alloc_grad固定buffer申请

resize动态分配和释放，导致显存碎片化和CUDA分配器开销风险

torch一般通过CUDA Caching Allocator向底层申请显存，而频繁申请和释放会导致显存碎片化，当某次再调用resize需要大块连续体的缓冲区时，发现总显存够，但是没有连续空间放得下，因此就需要出发empty_cache()释放空闲碎片，为了安全释放显存，GPU需要确保正在执行的kernel已经用完了这些内存，因此强制调用了cudastream synchronize导致cpu挂起，等待gpu。

实际案例：zero2_buffer_resize

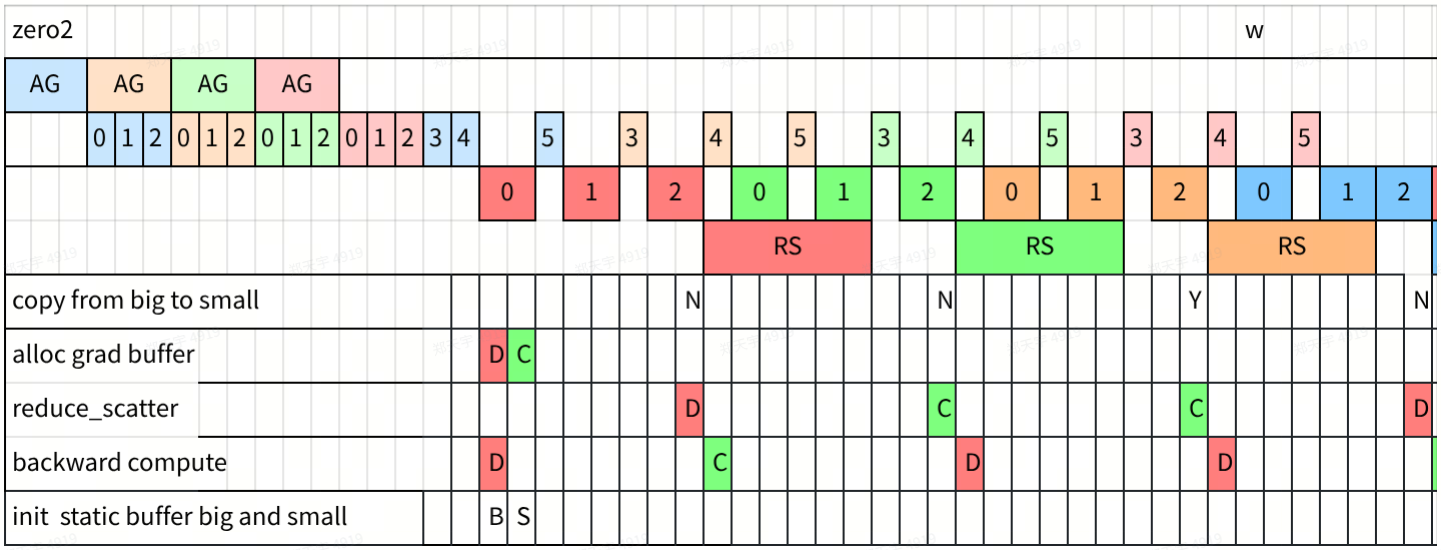
当前：



当采用grad 计算和通信overlap的时候，同一个stage上，两个chunk交替时，至少需要两个buffer，一个用于存计算完的grad做reduce_scatter通信，而另一个用于计算下一个chunk的backward的grad。

当前使用的是resize方法，即reduce_scatter做完后，会释放当前chunk的buffer，下一次需要时再申请，当chunk比较多的时候，频繁resize会导致显存碎片化严重，例如上述的问题，为了安全整理显存，GPU需要确保kernel用完了内存，强制调用cudastream synchronize，此时cpu挂起等待gpu。

现在：



第一次不需要copy，此时直接用big buffer做RS，而small buffer做backward compute；

第二次，此时本身就在small buffer做compute，所以下一个chunk放在big buffer上一定够，因此不需要copy；

第三次，发现下一个chunk比small buffer大，那么small buffer没办法用来存backward compute得到的结果了，因此要做一次copy。

第四次，此时下一个chunk比small buffer小，因此可以直接用small buffer存backward compute得到的结果，因此不需要copy。

现在的static buffer的逻辑是，由于不同chunk可能存在不同的模型层，比如lm-head，embedding，tranksformer block，这时候chunk之间需要的显存大小是不相等的，因此无法分配两个固定的buffer来给所有chunk使用，因此这两个buffer需要特殊制定。

Big buffer：为了满足每个时刻RS和compute可以overlap，因此我们必须保证一定有一个buffer能适用于所有chunk，当然最简单的实现方法是，big buffer*2，分别用来做RS和compute，但是这样容易浪费显存。

Small buffer：为了节省显存，我们其实只需要保证有一个buffer适用于所有chunk，两个chunk中参数小一点的用smaal buffer。

- 那么这个small buffer如何设计能满足所有chunk间的切换？

  其实只需要收集两两相邻chunk 参数最小的那一方，如下：

  pair_chunk_bytes:{ min(c1,c2),min(c2,c3),min(c3,c1)}。

  再取small buffer=max(pair_chunk_bytes)

  这样small buffer一定满足在任意相邻的两个chunk间，能装下较小的那个chunk的grad，而另一个则用big buffer一定能装下。

- 最后，当发现当前chunk在计算时是在big buffer上计算，且下一个chunk的参数比当前chunk大时，则需要做一次copy，从而满足两个chunk的grad都能被buffer装下。