

pp调度通信的buffer研究

背景：

分析当前的流水并行，fwd bwd 层间主要的影响来源于流水线的p2p暴露的时间：

通道	占用 s	被计算盖住 s	隐藏率	裸露 s	裸露占整步
PP p2p	13.91	8.11	58.3%	5.81	22.3%

Pp p2p暴露了5.8s，占比是22.3%，我们在优化之前得知道，这个百分比是正常还是异常，理论应该占比是多少，是否还有优化的空间？

这里以一个3dtimeline为例：



我们以 layer=88层，micro_batch=32，vpp=11，pp=8，来做计算：

以forward_send 为准，假设prefetch recv都提前发起了，fwd send和fwd compute时间大概是3:4。forward_send时间是3，则fwd_compute的时间大概是4，bwd就是8。

非通信bubble： $(t_{fwd}+t_{bwd}) * (pp-1) = (3+8)*7=77$

通信bubble： $t_{send} * ((vpp-1) * pp + pp - 1) * 2 = 3 * 8 * 2 = 486$

计算时间： $(t_{fwd}+t_{bwd}) * micro_batch * vpp = 12 * 32 * 11 = 2112$

总buble理想占比： $(77+486)/4787=11.7\%$

而现在p2p暴露是22.3%，理论看还不太符合预期。

这里也可以算一下send的带宽，一层是 $4 * 12288 * 4096 * 4 / 1024^3 = 0.75GB$ ，

$0.75 / 14.68 * 1000 = 51.09 GB/s$ ，带宽才打到一半。（这块测试的时候可能因为层数少，大部份是跟其它通信overlap导致的）

pp通信bubble研究：

这里引出一系列问题：

- 这个通信bubble怎么来的，跟什么相关？
 - 换成put only的方式能否解决这些通信bubble？
- 怎样可以减少通信bubble？
 - 理想的 pp，micro_batch，vpp的配比是什么样的？
 - 是否跟单个vpp chunk的fwd，bwd时间，以及send recv时间相关？

1. bubble来源

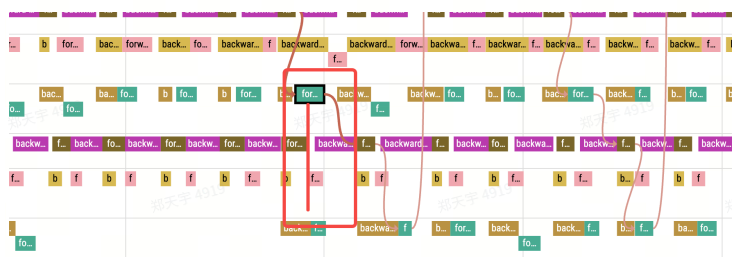
首先对于vpp来说，固定的fwd bwd 非通信bubble是因为warmup和cooldown阶段必然会产生产生的bubble，这部分免除不了，而通信bubble的原因可以看下图：（这里设置fwd_time=4,bwd_time=4,send_time=1）



warmup阶段，每个pp_stage计算完成后，都会有一段send通信的时间发送到下一个pp_stage上，这部分是在关键路径上的，因此引入了通信bubble。

而我们改成put only，从当前这个理论上来看，是减少不了整个pp的时间的。

不过3dtimeline上观察到下面的情况：即使我们用了profetch，但是从timeline上看还是会出现send比recv先发起，这时候就是send在等待fwd了。



链路	对数	recv 先起	send 先起	send 先起占比	send 领先 p50	真实共存段 p50
0→16 (fwd)	2240	1443	797	35.6%	5.41 ms	10.03 ms
16→32 (fwd)	2240	1786	454	20.3%	5.98 ms	10.46 ms
32→48 (fwd)	2240	865	1375	61.4%	5.78 ms	10.93 ms
48→0 (fwd)	2128	999	1129	53.1%	5.89 ms	10.75 ms
48→32 (bwd)	2240	1985	255	11.4%	12.67 ms	11.44 ms
0→48 (bwd)	2128	1726	402	18.9%	7.24 ms	11.71 ms
16→0 (bwd)	2240	1382	858	38.3%	13.12 ms	11.90 ms
32→16 (bwd)	2240	1004	1236	55.2%	13.58 ms	11.00 ms

这块引入两个可能的优化点在于：

- 如果send改成单边put，这里通信提前完成，一方面可以让出带宽，给overlap的通信，另一方面可以让出SM给overlap的计算，理论上还是有收益的。

- 改成单边put，就可以观察到，实际的编排和理想编排具体哪里慢了，因为理想编排按提前recv，其实就是单边put。

2. 怎样可以减少通信bubble?

先来看一个比较简单的例子，还是上面的例子，我们可以看到这个条件下，这里的bubble其实可以用micro_batch来填充，并且只需要一个micro_batch。



如上面右图所示，如果实际运行的时候，fwd，bwd，fwd_send时间如上图所示，这里新增一个microbatch即可覆盖住通信的bubble。除了最开始gpu空转的bubble。

这个例子比较简单，fwd和bwd时间相同，且fwd刚好是fwd_send的pp倍数，而实际运行的时候，这里配置相对复杂。

盖住通信bubble公式化：

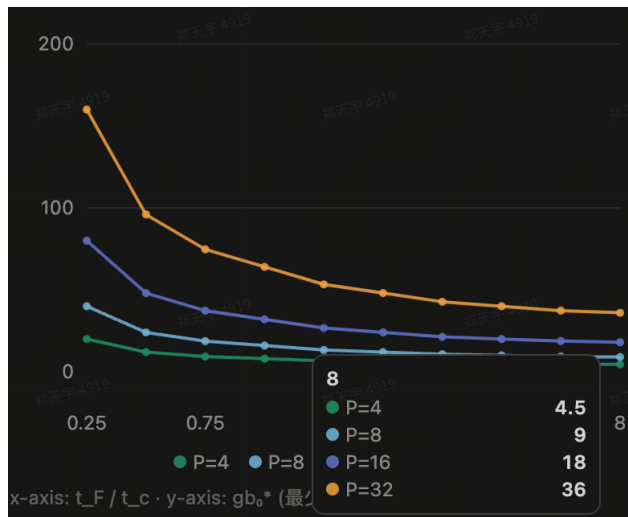
满足如下公式：

$$gb_0 \cdot t_F \geq P \cdot (t_F + t_c) \rightarrow (gb_0 - P) \cdot t_F \geq P \cdot t_c$$

这里即第一个group的微批次数量做fwd的时间，超过第一个micro_batch传回来的时间，可以算一下，按我们现在线上的配比：

$t_F=4$ ， $t_c=3$ ， $P=8$ ，则 gb_0 至少 ≥ 14 ，即从8增加到14，而显存从88->154, 418G->731.5G，才能盖住。

这里: 可以看到，纵轴是需要的 gb_0 大小，横轴是 t_F/t_c ，当 t_F/t_c 比例达到6时，此时 $pp=8$ 时，只需要增加一个micro_batch即可，所以其实可以增加vpp_chunk的层数。，一方面降低整个 t_c 占比，另一方面也可以减少micro_batch的增加就能把通信bubble盖住。其它地方之前讨论也可能需要，比如用上vwn优化？



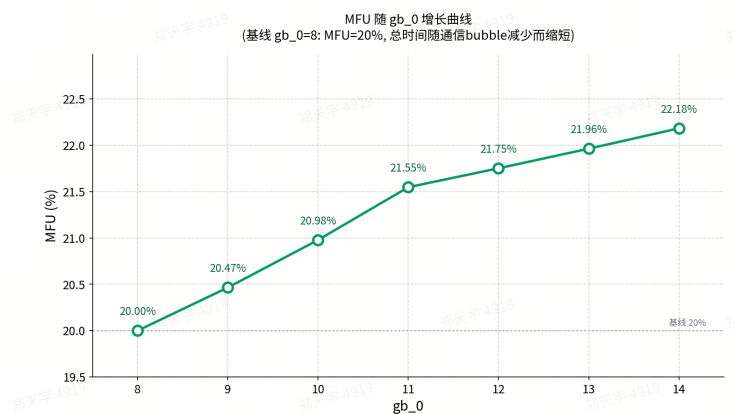
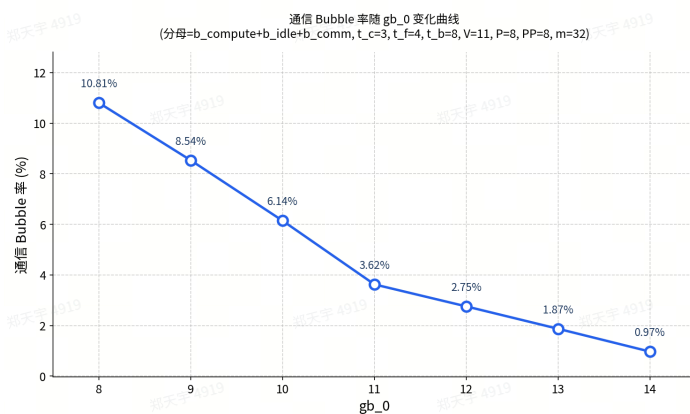
若不满足，则转换成公式即（bwd与fwd对称，这里就只写fwd的时间了）：

$$B_{warmup} = (V - 1) \cdot \max(0, P \cdot t_c + (P - gb_0) \cdot t_F) + (PP - 1) \cdot t_F$$

因为Any1F1B要求，gb0一定大于等于P,所以后面一项一定是负的，同样的结论，tF和tc比值越大，这里的通信bubble越小。

可以看到计算时间tF：通信时间tc，比值在8时，此时基本可以（或增加1个micro_batch）保证没有通信bubble。e2e理论降低11%的时间。

同时我们也可以分析，在当前状态下，增加micro_batch能降低的bubble率和增长的mfu是多少：



3. 怎样可以不增加gbs来调整这个分组的micro_batch编排？（因为gbs一般是算法定好的，定版后不能随意更改了）

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23	T24	T25	T26	T27	T28	T29	T30	T31	T32	T33	T34	T35	T36	T37	T38
Rank 0	F0	F1	F2	F3	F4	F0	F1	F2	F3				B0	F4	B1	F5	B2	F6	B3	F7	B4	F5	B0	F6	B1	F7	B2		B5		B6		B7	B3	B4	B5	B6	B7	
Rank 1		F0	F1	F2	F3	F4	F0	F1	F2				B0	F3	B1	F4	B2	F5	B3	F6	B4	F7	B0	F5	B1	F6	B2	F7	B5		B6		B7	B3	B4	B5	B6	B7	
Rank 2			F0	F1	F2	F3	F4	F0	F1				B0	F2	B1	F3	B2	F4	B3	F5	B4	F6	B0	F7	B1	F5	B2	F6	B5	F7	B6		B7	B3	B4	B5	B6	B7	
Rank 3				F0	F1	F2	F3	F4	F0	B0	F1	B1	F2	B2	F3	B3	F4	B4	F5	B0	F6	B1	F7	B2	F5	B5	F6	B6	F7	B7	B3	B4	B5	B6	B7				

如上，逻辑上和 min memory 处理余数的逻辑差不多

4. 哪些因素可以降低pp通信bubble?

$$R_{bubble} = \frac{\overbrace{(p-1)(t_f + t_b)}^{B_{idle}} + \overbrace{2t_c(pv-1)}^{B_{comm}}}{\underbrace{vm(t_f + t_b)}_{T_{calc}} + \underbrace{(p-1)(t_f + t_b) + 2t_c(pv-1)}_{B_{total}}}$$

这里我们来做一下理论分析，这其实是一个数学建模的问题，当 t_f , t_b , t_c 固定，我们可以扫描出场景下满足约束条件的最小bubble组， (v, p, m)

我们现在就简单直观来看：

1. 增加micro_batch数，一定能降低bubble。
2. 增大 $(t_f + t_b)$ ，可以看到虽然， $(t_f + t_b)$ 会增大空转bubble的占比，其实可以发现， $(t_f + t_b)$ 的增大，对空转bubble率并不会增加，但是会稀释通信的bubble
3. 降低 t_c ，这个就是直接减少通信bubble的占比
4. bubble率的两个分子都带 p ，而分母的计算时间不带 p ，所以我们除以 p ，可以发现， p 开的越大，分母越小，bubble率也越高

所以结论是， m 增大， t_f, t_b 增大，降低 t_c ，减少pp并行度就是我们可以减少ppbubble的方向。

5. 什么时候megatron和fsdp性能相等?

均切micro_batch_nums显存放不下：

单batch 总token： $\text{micro_batch_nums} * \text{micro_batch_len}$

单micro_batch放不下两种解法：

1. 切micro_batch_len，即开大sp，无法对sample做进一步切分，存在padding的overhead。

2. 增大micro_batch_nums，单个micro_batch_len降低



micro_batch_nums 切成4时，单个micro_batch放不下。切成7份，4张可用gpu：

fsdp：2轮时间，即2T

megatron：(T/3/4)*((4+3)*3+(4-1))=2T

这里补充一点megatron的优势：

megatron可以更好的overlap 通信，如图，f0-f6都用的是同一层参数做fwd，他们就可以用同一份参数，第二层黄色部分的参数，有7个micro_batch计算做overlap，而对于fsdp来说，就是只有一个micro batch做overlap。