LLM训练中的一些数字

本文主要探讨近年来一些主流的显存优化技术和并行技术对算力配置的影响，旨在帮助大家评估既定平台和目标模型下的分布式训练开销，同时了解训练过程中的通信计算模式。欢迎大家讨论指正。

基本信息

在计算开始前，首先我们需要了解一些主流GPU型号和模型的相关参数，这能够帮助我们估算训练过程中的一些重要指标。

首先是GPU：

	Tensor FP16 算力 (Tflops)	FP32 算力 (Tflops)	内存容量(GB)	内存带宽 (GB/s)	通信带宽 (GB/s)	通信时延 (us)
H200	989	495	144	4900	900	1
H100	989	495	80	3350	900	1
A100	312	156	80	2000	900	1
4090	330	83	24	1000	64	10
3090	142	36	24	936	64	10

然后是一些主流的LLM模型：

	token长度(s)	head num(a)	hidden layer size(h)	层数(L)	词汇表大小(V)
LLAMA-7B	4096	32	4096	32	32000
LLAMA-13B	4096	40	5120	40	32000
LLAMA-32B	4096	52	6656	60	32000
LLAMA-65B	4096	64	8192	80	32000
LLaMA2 70B	4096	64	8192	80	32000
Gpt3	2048	96	12288	96	50257
GPT-3 Small	2048	64	768	12	50257
GPT-3 Medium	2048	64	1024	16	50257
GPT-3 Large	2048	96	1536	16	50257
GPT-3 XL	2048	128	2048	24	50257
GPT-32.7B	2048	80	2560	32	50257
GPT-3 6.7B	2048	128	4096	32	50257
GPT-3 13B	2048	128	5140	40	50257
GPT-3 175B	2048	128	12288	96	50257

第一步：评估模型参数量

参数量(φ)是一个模型的重要指标，事先了解模型的参数量可以帮助我们快速估计一些更复杂的指标，比如计算量和显存占用。

一般来说评估tranformer模型的参数量我们会考虑以下几个组分，这里分别给出它们相应的计算公式：

- Word embedding: $h \times v$
- Self attention: $4 \times h \times h$
- Feed forward: $8 \times h \times h + 5 \times h$
- Position embedding: $h \times s$

把这些组分全部包含进来，可以得到模型总的参数量公式：

$$\varphi = h \times (v + s) + (12 \times h \times h + 5 \times h) \times L$$

不同类型的模型（比如GPT和LLaMA）在部分系数上会有一些区别，我们这里主要以GPT系列的模型为例

第二步：选择合适的并行度

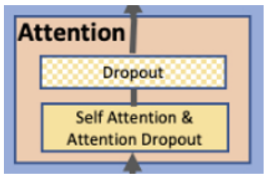
考虑流水线并行

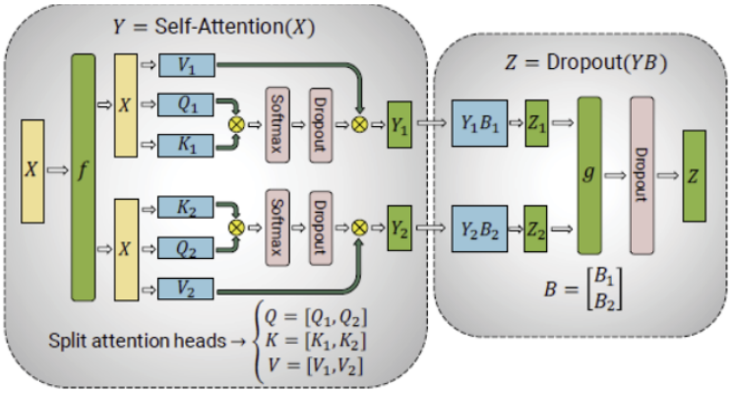
现在常用的流水线并行算法叫做PipeDream-flush，其调度方法如图所示，具体的技术细节可以参考论文。

流水线并行的一大问题是正向传播的activation需要在2*（k-1）个microbatch后的后向传播中用到（这里的k是当前设备在流水线中从后往前数的位次），因此所有activation会随着流水线运转堆积，流水线越长上游设备需要保存的activation越多（我们称流水线并行度为p）。因此虽然流水线并行减少了模型的层数，但是对于上游设备来说，activation的总大小并不会随着流水线并行度的增加而减少（$L/p * p = L$）。由于activation和显存需求的详细计算需要考虑张量并行，而显存需求又会反过来影响流水线并行度，因此我们留到后续讨论。

其次，我们需要足够多的microbatch数量和时间来填满流水线，对于长度为p的流水线，我们的microbatch数量m至少要大于p，且填充流水线所消耗的bubble time为(p-1)/m，为了让bubble time的占比足够小，通常会把m设置在p的4倍以上。然而，模型最终的batch size (B)等于数据并行度（d）乘以梯度累积步数（gradient accumulation，g）乘以microbatch数量(m) 乘以microbatch大小(b)，亦即$B = d \times g \times m \times b$，因此过长的流水线步数同样会导致B过大。

考虑张量并行





张量并行对模型的层内结构进行了更细致的切分，我们可以把多个attention head拆分到不同的GPU上。

每个 head 里面的 Q、K 两个矩阵的大小是 minibatch size * token 长度 * key 的大小，V 矩阵的大小是 minibatch size * token 长度 * value 的大小（注意，这只是一个 head 的）。key/value 的大小一般等于 hidden layer size / heads 数量。而 Q、K、V 参数矩阵在每个 head 上的大小是 hidden layer size * hidden layer size / head num 。

那么每个head在正向传播中的计算量和通信量分别是多少呢?

- 计算量：2 * 3 (Q, K, V) * minibatch size * token 长度 = 2 * 3 * B * s * h * h / a
- 通信量：minibatch size * token 长度 * hidden layer size / heads num * bytes per param = B * s * h / a * 2

这里我们需要尝试引入张量并行（t）以保证计算时间和通讯时间尽可能保持平衡，这有利于我们将计算和通信进行重叠：

令$h' = h/t$，通信时间≤计算时间，我们得到$t \leq 3 \times h \times$ 单工内存带宽 / GPU算力

可见不同类型的GPU能够支持的模型并行度实际上受限于GPU本身的算力和通信带宽。一般来说为了保证NVLink可用，模型并行通讯被限制在单机内进行，这意味着实际的模型并行度不会超过8。

回过头再看显存挑战

对于一个已知参数量的模型来说，直接显存开销包括以下几个部分：

- Fp16 weight for train-2φ
- Fp16 grad for train-2φ
- Fp32 weight for optimization-4φ
- Fp32 momentum for Adam-4φ
- Fp32 variance for Adam-4φ

一共是16φ字节的显存开销，注意到不管是流水线并行还是张量并行都会减小模型的参数量，因此每张卡上实际的直接显存开销为$2shBL/t$。

除此之外，还要考虑需要缓存的正向传播过程中产生的activation： A = 层数*token 长度 * microbatch size * hidden layer 的神经元数量 * （34+5 * head num * token 长度 / hidden layer size）= $shBL(34+5as/h)$

注意流水线并行并不会改变activation的大小，因为假设总层数为L，每个GPU需要保存L/p层，同时对于位于第一个stage的GPU来说它需要缓存p个这样的activation，因此总的activation仍为$L/p*p=L$。

但是前面提到过张量并行可以减小h，因此能够减小activation的大小：
$A=shBL(10+24/t+5as/h)$（注意这里张量并行并不作用于transformer的每一组分）

到这一步为止我们已经可以推出模型训练所需的最小流水线并行度：$16\varphi/pt+A\leq M$，其中M为GPU 可用显存，亦即$p\geq 16\varphi/t(M-A)$

可以发现对于显存来说最大的挑战仍然是activation部分，因为流水线并行并不能很好地减少这部分开销。

当然，到目前为止有很多方案可以压缩这个显存消耗，比如Reducing Activation Recomputation in Large Transformer Models这篇文章提到的sequantial parallelism + Selective Activation Recomputation技术可以将activation压缩到：层数*token 长度 * hidden layer size * 34 / 张量并行度 =$34shBL/t$，Full recomputation技术可以把activation压缩到：$2shBL/t$

第三部：考虑训练过程中发生了哪些通信和计算

首先我们要知道在一个iteration的训练周期当中，GPU上发生了哪些通讯事件。

首先，在每个iteration的末尾，我们需要收发每台机器上的8张GPU产生的梯度。实际上由于模型并行的存在，这些GPU会被分成几个小组进行独立的allreduce，但这不妨碍每个GPU都需要在每个iteration末尾进行一次梯度聚合。实际进行数据并行通讯的时候，每个GPU需要传输的数据量取决于其实际持有的模型大小。在显存挑战一节中，我们讨论过三种不同的并行配置，它们将模型切分成不同大小的分片，这里我们只关心pt的乘积，它决定了每个GPU上的参数量。由于梯度传输用的是fp16精度，因此我们需要传输$2\varphi/pt$的数据到网络。通常来说8张GPU会共享服务器的整个网络，因此一共需要传输$2shBL/t$字节的梯度，再考虑到目前主流的ring allreduce算法会带来一倍的通讯开销，也就是一共$32\varphi/pt$字节的传输量。

其次，在流水线的调度过程中，每张GPU会在完成一个microbatch的计算后立刻将activation/gradient传给相邻的rank，这里用到的是流水线并行组中的p2p传输，传输量为$2 * h * h * B/t * 8 = 16Bh^2/t$。在实际训练过程中这部分通信开销通常会与计算重叠。

此外，在每个microbatch的计算过程中，每个GPU上的子模型还会发生张量并行通讯，对于每个transformer层的每个线性层，这样的通讯包括：

- forward计算前的allgather：$2 * h * h/t * b * t = 2bh^2$
- backward计算中的allgather：$2 * h * h/t * b * t = 2bh^2$
- backward计算中的reduce-scatter：$2 * h * h/t * b * 2 = 4bh^2/t$

需要注意的是，backward过程中的通讯可以与部分计算重叠（这也是我们需要谨慎考虑张量并行度的原因之一），而forward过程中的则不行。

这样的通讯每个microbatch在每个transformer层一共发生四次，每个GPU上一共有L/p层，共有m个microbatch，

因此总通讯量为：$16bh^2(1 + 1/t)Lm/p$

可以看到这是相当大的一笔通讯，因此通常来说我们只把张量并行组设置在同一个机器内，利用GPU通信带宽进行传输。

而对于计算量的估计则相对简单，已知每张卡上的参数量φ/pt，我们可以认为forward和backward计算的对应开销为：

- forward：$2sB\varphi/pt$
- backward：$4sB\varphi/pt$

有了这些指标以后，我们可以很方便地画出GPU在每个训练周期内的计算-通信图。