深度学习技术及应用
国家工程研究中心 | 飞桨

自动并行下vpp流水并行策略的功能增强

答辩人：郑天宇 / zty-king

指导人：陈锐彪 / From00

飞桨护航计划集训营

自我介绍：郑天宇

>>>>>>>>

我是电子科技大学计算机专业的研一学生，研究方向包括大模型、计算机视觉、深度学习。9月初，我首次接触飞桨，满怀激情，决心加入其中，力争有所作为。本次参与护航计划，我对飞桨平台有了深入的了解，在模型分布式训练—流水并行编排方向进行了大量的实践。

>>>>>>>>>

PART1：开发任务简介与个人拆解

PART2：实际开发工作介绍

PART3：未来工作规划

PART4：总结与体会

PART1 ：开发任务简介与个人拆解

>>>>>>>>

任务1：自动并行下的非均衡vpp并行策略研发

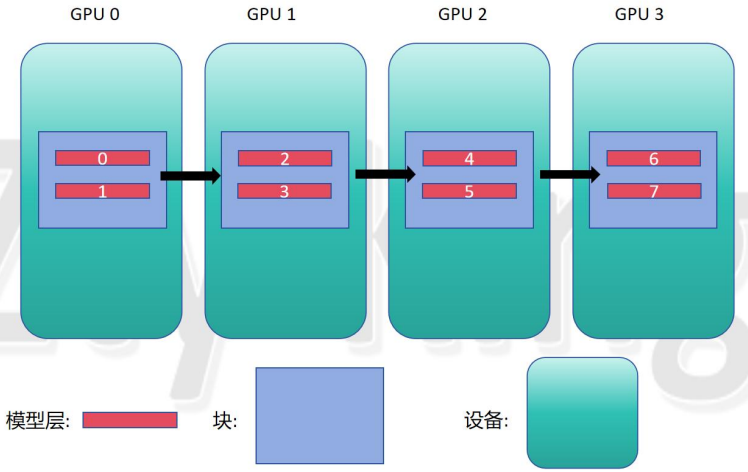
- 支持acc_step % pp_degree != 0的情况
- 支持hidden_layer % num_chunks != 0情况（num_chunks=pp_degree*vpp_degree）

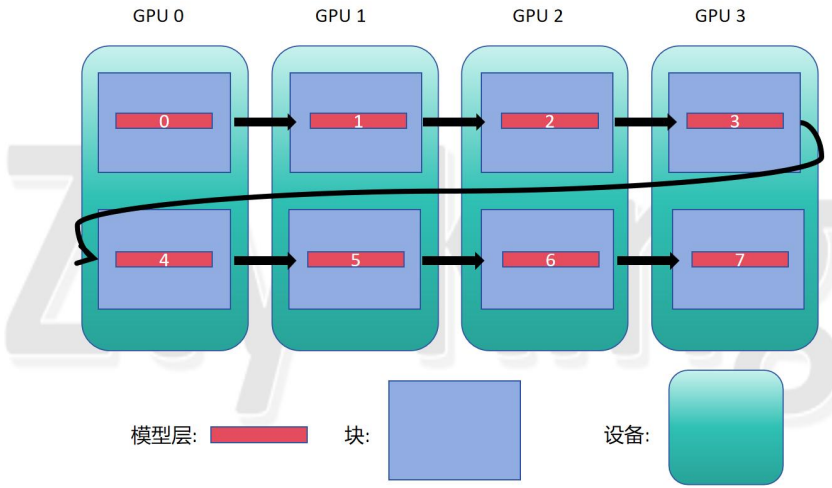
任务2：在线aadiff检查工具调研和方案设计

- 公式化bubble在各种流水并行编排模式下出现的位置
- 在各编排模式下的bubble期用抽象方法检测异常

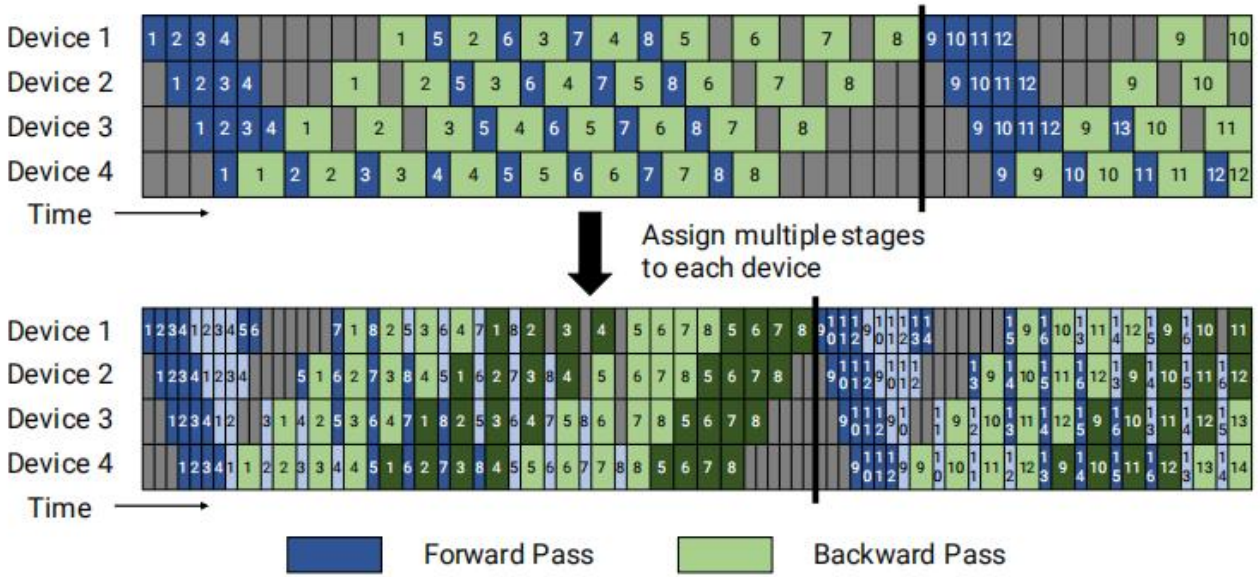
PART2：实际开发工作介绍-vpp流水并行介绍

重切分阶段





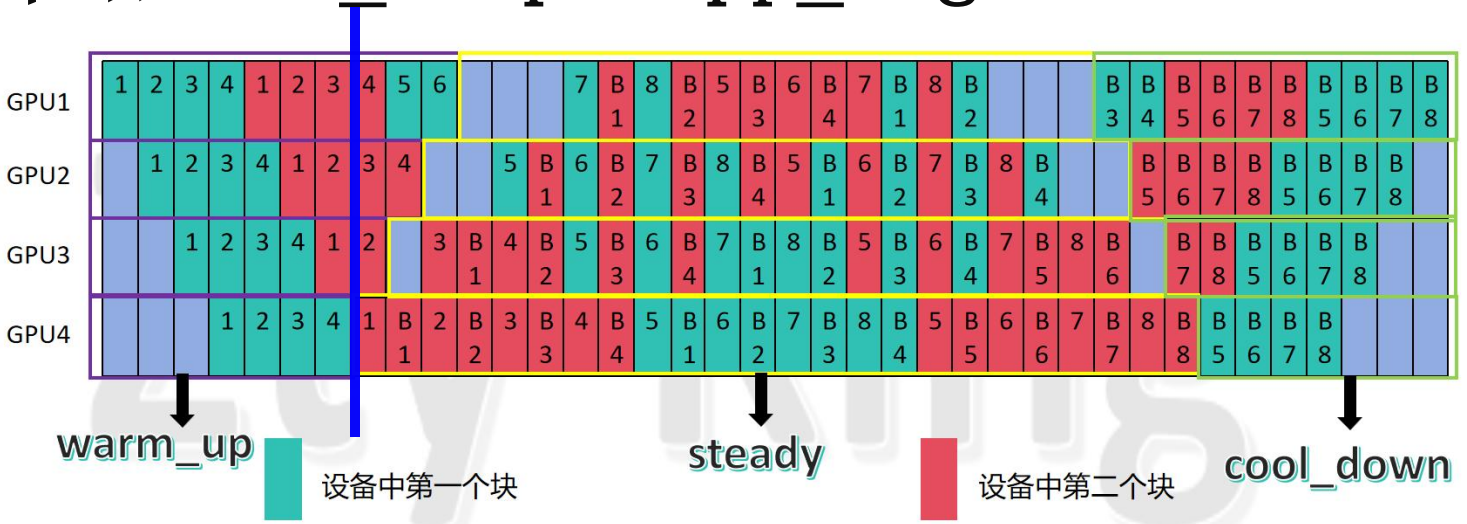
编排阶段



PART2：实际开发工作介绍-acc_step % pp_degree != 0

>>>>>>>>>

均衡vpp编排下的Step分析



pp_degree=4,acc_step=8,vpp_degree=2

- 通用公式总结：

1. 计算 warm_up_steps ：

$$\text{warm\_up\_steps} = 2 \times (\text{pp\_degree} - \text{pp\_stage} - 1) + (\text{vpp\_degree} - 1) \times \text{pp\_degree}$$

2. 计算 steady_steps ：

$$steady\_steps = acc\_steps * vpp\_degree - warm\_up\_steps$$

3. 计算 cool_down_steps：

与warm_up对称，因此公式一致

PART2：实际开发工作介绍-acc_step % pp_degree != 0

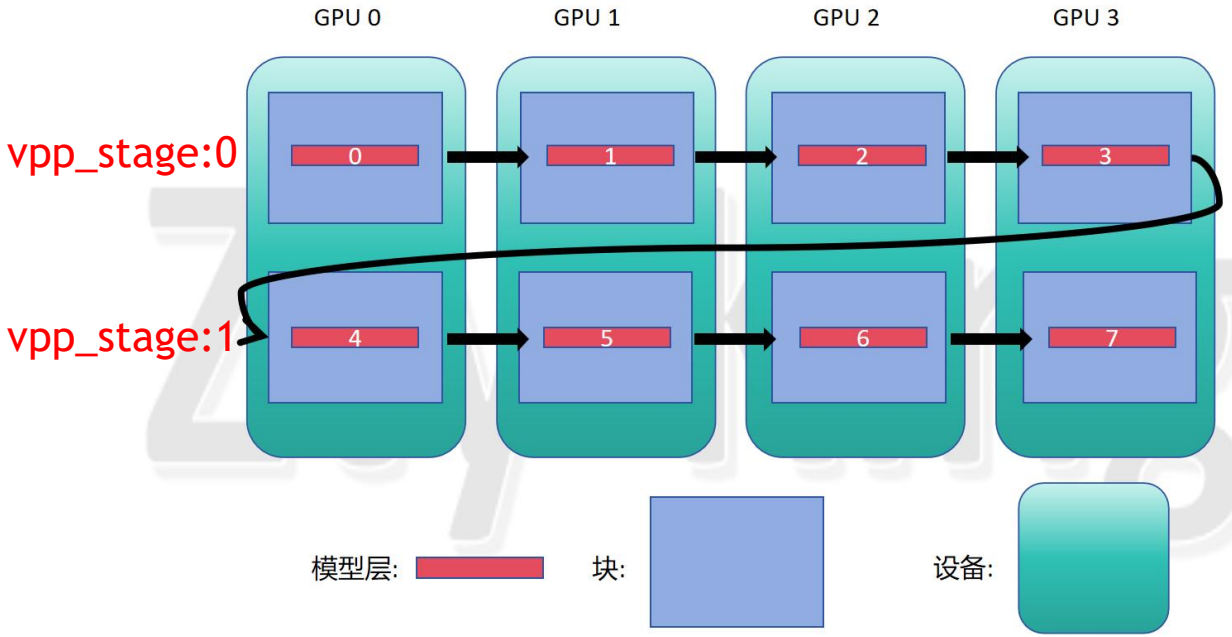
>>>>>>>>>

均衡vpp下的单个设备的job编号编排分析

- 重要公式：(仅看foward，backward对称)

计算 vpp_stage（即对应块号）：

$$\text{vpp\_stage} = \left\lfloor \frac{(\text{micro\_step} \,\%\, (\text{pp\_degree} \times \text{vpp\_degree}))}{\text{pp\_degree}} \right\rfloor$$

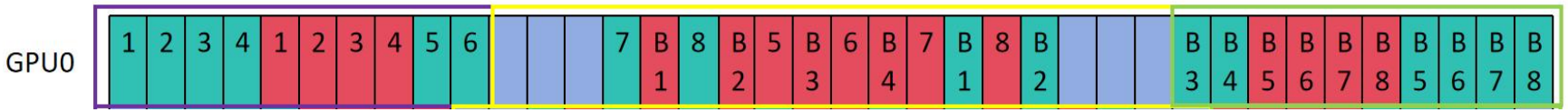


vpp_degree：2

pp_degree：4

pp_degree=4,acc_step=8,
vpp_degree=2,hidden_layer=8

micro_step: 0 1 2 3 4 5 6 7 8 9



vpp_stage: 0 0 0 0 1 1 1 1 0 0

job编号: f_00 f_02 f_12 f_05

PART2：实际开发工作介绍-acc_step % pp_degree != 0

>>>>>>>>>

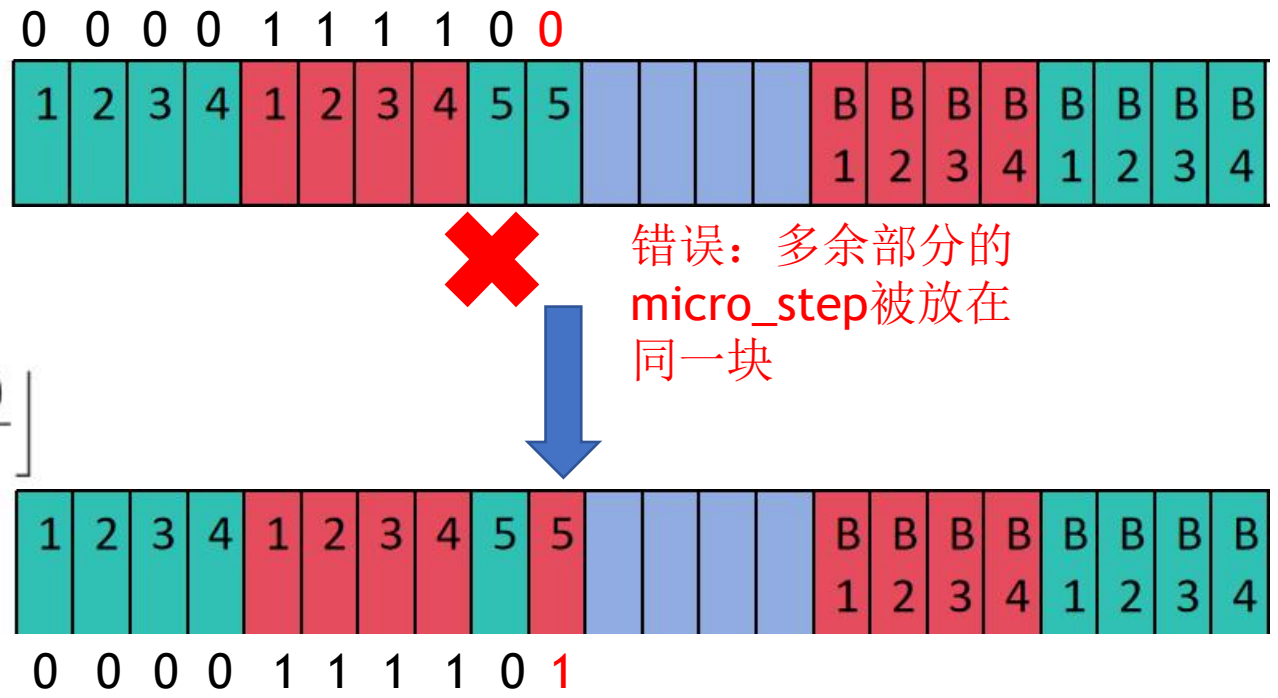
非均衡vpp下的单个设备的job编号编排分析

（step显然成立，不再分析）

- 验证公式：（仅看核心公式计算vpp_stage）

$$\text{vpp\_stage} = \left\lfloor \frac{(\text{micro\_step} \mod (\text{pp\_degree} \times \text{vpp\_degree}))}{\text{pp\_degree}} \right\rfloor$$

第9和第10个micro_step的vpp_stage为：
vpp_stage_9=8 mod 2*4 // 4 =0
vpp_stage_10=9 mod 2*4 // 4 =0
均被分到了第0层，因此原来的公式不再适配



pp_degree=4,acc_step=5,vpp_degree=2

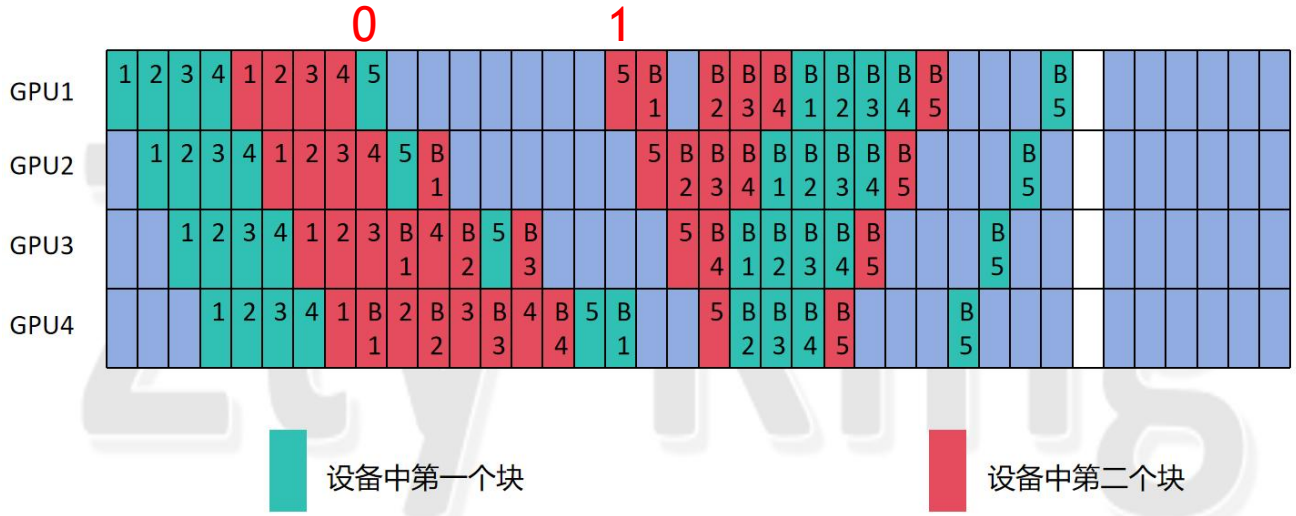
PART2：实际开发工作介绍-acc_step % pp_degree != 0

>>>>>>>>>

非均衡vpp支持并与均衡VPP统一

- 定位冲突位置

1.分析可知，出错的地方主要在acc_step与pp_degree的余数部分，因此我们对余数单独处理：



pp_degree=4,acc_step=5,vpp_degree=2

```
remainder = acc_steps %  pp_degree
vpp_stage = micro_step % (pp_degree * vpp_degree)
if micro_step <= (acc_steps // pp_degree)*(pp_degree * vpp_degree):#整除部分，非余数部分
    vpp_stage = vpp_stage // pp_degree
else:余数部分
    vpp_stage = vpp_stage // remainder
if not forward:
    vpp_stage = vpp_degree - vpp_stage - 1 #backward为forward在模vpp_degree下的相反数
return vpp_stage
```

代码同时适配均衡VPP与非均衡VPP

PART2：实际开发工作介绍-acc_step % pp_degree != 0

>>>>>>>>>

非均衡vpp性能分析与优化

- 在论文基础上改进的编排方法，即按照每pp_degree个forward轮巡编排，则最后余数部分最后处理，如右边上图所示，此时由于最后一个micro_batch，将产生11个bubble
- 本次工作探索发现，非均衡vpp仍然可以进行对称编排，并且按照对称编排后，bubble率将和均衡vpp保持一致，

！bubble率：从26%降低到14%
极大提升了效率

论文基础改进的非均衡编排方法

本次工作探索的非均衡编排方法

PART2：实际开发工作介绍-hidden_layer % num_chunks != 0

>>>>>>>>

当前vpp的reshard分析：

- 由于当前vpp流水并行仅支持hidden_layer % num_chunks的reshard，主要原因在于当前假定：

  hidden_layer % num_chunks == 0

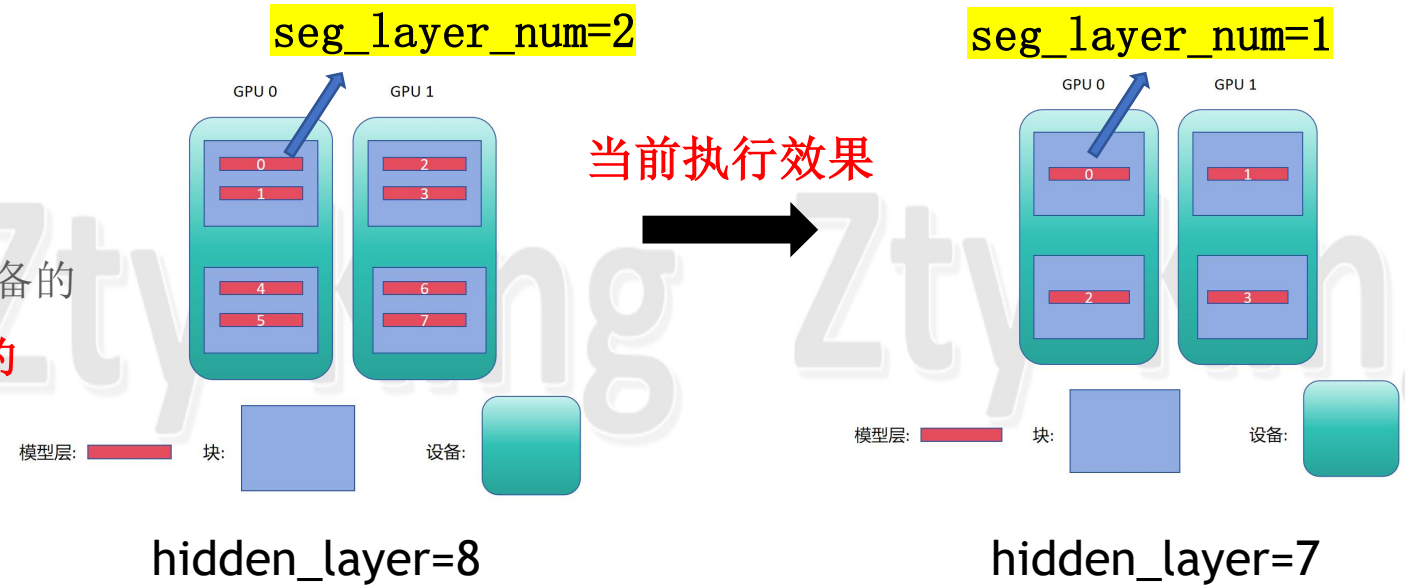
  seg_layer_num为一个定值

- 当模型层数减少一层时，会导致每个设备的每一个块的seg_layer_num均变为原来的seg_layer_num-1，导致出错

seg_layer_num表示一个块中的layer数

```
seg_layer_num = len(seg_struct_names) // num_chunks
for seg_id in range(num_chunks):
    start_idx = seg_parts[seg_id * seg_layer_num]
    end_idx = seg_parts[seg_id * seg_layer_num + seg_layer_num]
    pp_stage = seg_pp_stages[seg_id]
    chunk_id = seg_chunk_ids[seg_id]
    struct_name = ",".join(
        seg_struct_names[
            seg_id * seg_layer_num : seg_id * seg_layer_num + seg_layer_num
        ]
    )
    process_mesh = sub_process_meshes[pp_stage]
```

相关核心代码



hidden_layer=8　　hidden_layer=7

PART2：实际开发工作介绍-hidden_layer % num_chunks != 0

>>>>>>>>

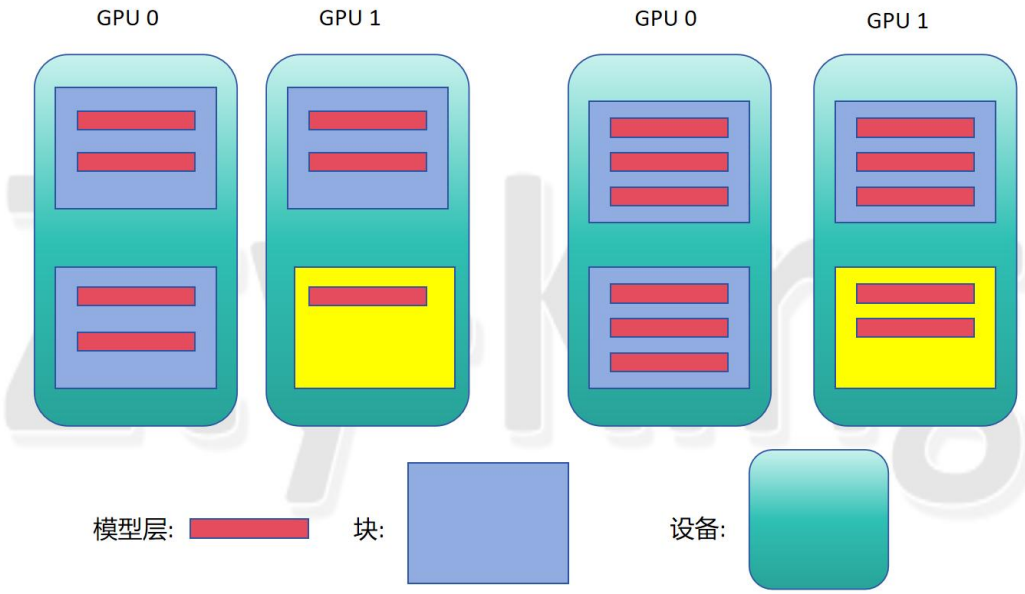
vpp去尾支持设计：

- 出错原因主要在于seg_layer_num，最主要问题是其被设置成了定值，因此此处采用列表来表示seg_layer_num，即每一个块对应一个seg_layer_num的值（也为后续工作奠定基础）

```
seg_layer_num=[0]*num_chunks#记录每个块里面的层数
    for j in range(0,len(seg_struct_names)):#把层数分别分给每个块，保证每个块至少先被分一层，按此逻辑可以支持少多层，只要保证每个块至少一层
        i=j%num_chunks
        seg_layer_num[i]=seg_layer_num[i]+1
```

```
    previous_seg_parts_end_idx=0#记录上一个终止点对应的seg_parts的index
    for seg_id in range(num_chunks):
        start_idx = seg_parts[previous_seg_parts_end_idx]#seg_layer_num需要改成对应块的层数，即用seg_layer_num[seg_id]代替
        end_idx = seg_parts[previous_seg_parts_end_idx + seg_layer_num[seg_id]]
```

代码同时适配均衡VPP与非均衡VPP



vpp去尾下，reshard后各模型层在块中的摆放

PART2：实际开发工作介绍-hidden_layer % num_chunks != 0

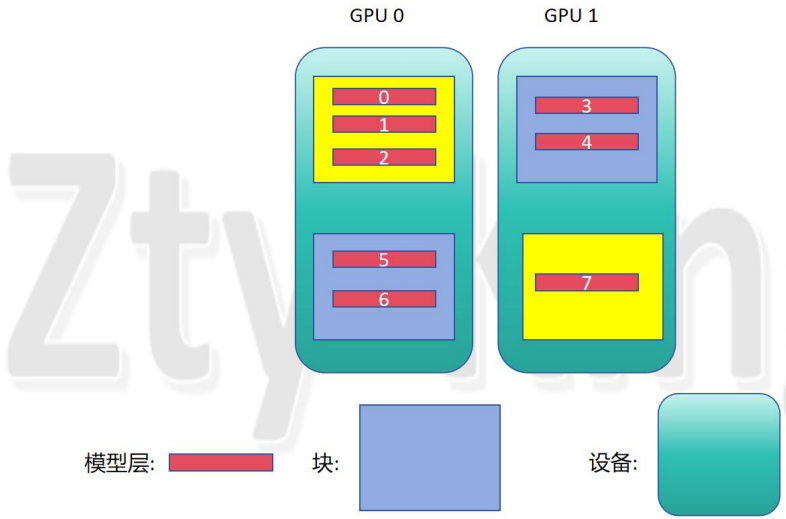
>>>>>>>>

基于vpp编排的灵活模型层分配策略研发：

- 使用vpp编排时可以任意指定各设备模型层数（保证每个块内至少一层）
- 当前代码分析

原来的代码无法正确编排，没有获取用户需求，按照用户实际需求分配

```
seg_layer_num = [0] * num_chunks
for j in range(0, len(seg_struct_names)):
    i = j % num_chunks
    seg_layer_num[i] = seg_layer_num[i] + 1
seg_parts = [0]
```



用户指定GUP0放置5层layer，GPU1放置3层layer

```
struct_name=[MyLinear,MyLinear_1],process_mesh=[{shape: [1], process_ids: [0], dim_names: [pp]}]
struct_name=[MyLinear_2,MyLinear_3],process_mesh=[{shape: [1], process_ids: [1], dim_names: [pp]}]
struct_name=[MyLinear_4,MyLinear_5],process_mesh=[{shape: [1], process_ids: [0], dim_names: [pp]}]
struct_name=[MyLinear_6,MyLinear_7],process_mesh=[{shape: [1], process_ids: [1], dim_names: [pp]}]
```

实际的分配方式

PART2：实际开发工作介绍-hidden_layer % num_chunks != 0

>>>>>>>>>

基于vpp编排的灵活模型层分配策略研发：

➢ 方法设计：

1. 获取用户的切分意图

2. 确保每个设备的分配层数大于等于块数

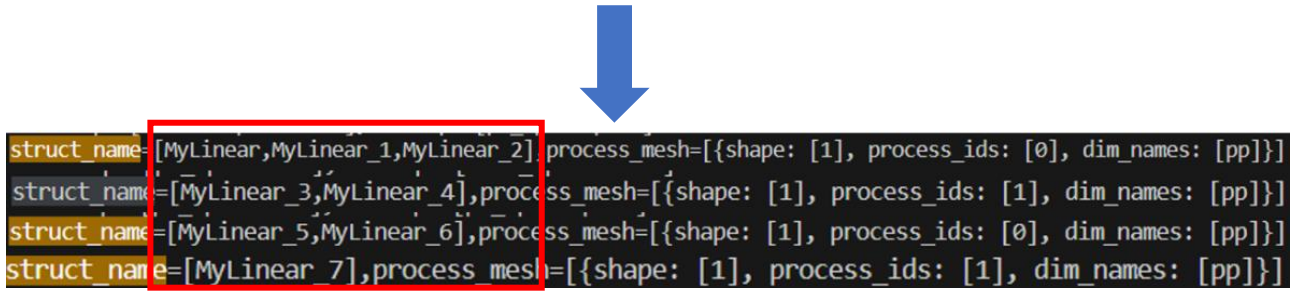
3. Round-Robin算法，对每个设备来说：设备中每个块轮循增加layer数，直到达到当前设备的指定数

4. get_user_layer_to_mesh为获取用户分配意图的函数

代码同时适配均衡VPP与非均衡VPP

```
user_layer_to_mesh = get_user_layer_to_mesh(
    ops, seg_method, pp_degree, len(seg_struct_names)
)
pp_stage_layer_num = [0] * pp_degree
for i in user_layer_to_mesh:
    pp_stage_layer_num[i] = pp_stage_layer_num[i] + 1
assert all(value >= vpp_degree for value in pp_stage_layer_num)
seg_layer_num = [0] * num_chunks
for pp_stage in range(0, pp_degree):
    pp_stage_layer_nums = pp_stage_layer_num[pp_stage]
    for i in range(0, pp_stage_layer_nums):
        v_chunk_id = i % vpp_degree
        r_chunk_id = (v_chunk_id) * pp_degree + pp_stage
        seg_layer_num[r_chunk_id] = seg_layer_num[r_chunk_id] + 1
seg_parts = [0]
```

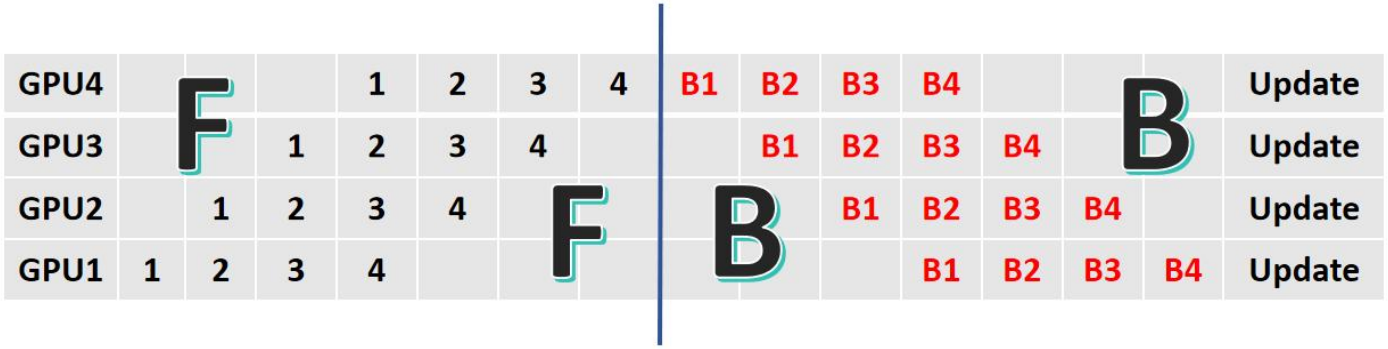
核心代码



最终效果

PART2：实际开发工作介绍-在线aa_diff检查

>>>>>>>>>



FthenB编排流水并行图

FthenB编排策略

- bubble出现在：

1. forward过程**编号高**的设备等待**编号低**的设备，处理完最后一个micro_batch后相反等待

2. backward过程**编号低**的设备等待**编号高**的设备，处理完最后一个micro_batch后相反等待

- bubble对应公式如下：

t_f即一个forward的时钟周期
t_b即一个backward的时钟周期

1. 在forward编排前，bubble个数为：pp_stage*t_f

2. 在forward编排后，bubble个数为：(pp_degree-pp_stage-1)*t_f+(pp_degree-pp_stage-1)*t_b

3. 在backward编排后，bubble个数为：pp_stage*t_b

PART2：实际开发工作介绍-在线aa_diff检查

>>>>>>>>>

1F1B、vpp编排策略

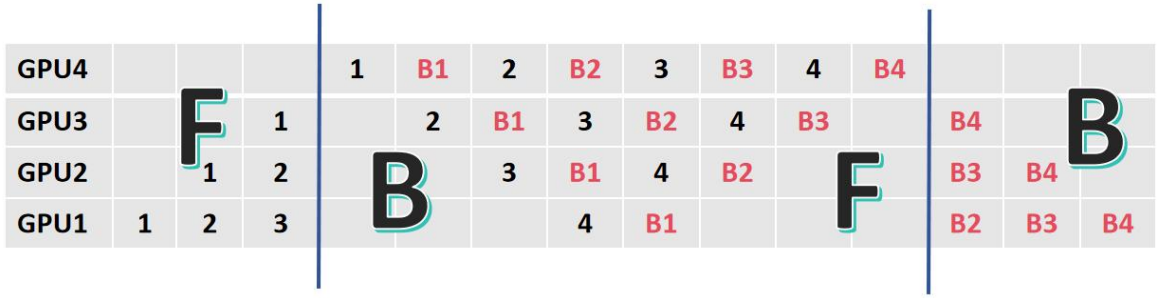
bubble对应公式如下：

1. 在warm_up编排前，bubble个数为：pp_stage*t_f
2. 在stabilization编排前后，bubble个数**分别为**：(pp_degree-pp_stage-1)*t_f、

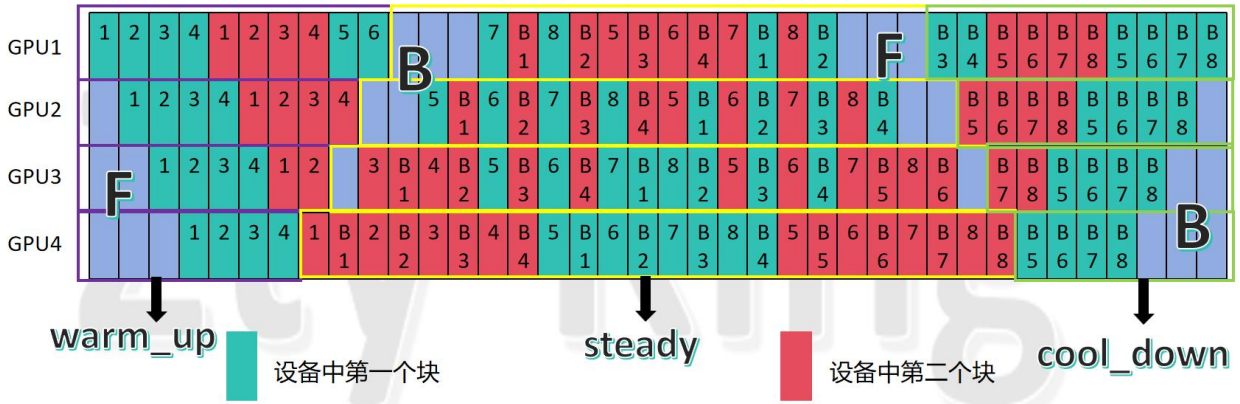
(pp_degree-pp_stage-1)*t_b

3. 在cool_down编排后，bubble个数为：pp_stage*t_b

➢ 1F1B结论：只改变峰值显存，不会降低流水并行时间——不改变bubble率和bubble时间



1F1B编排流水并行图



1F1B编排流水并行图

PART2：实际开发工作介绍-在线aa_diff检查

>>>>>>>>>

以1F1B为例做可行性验证

- 不同条件下的公式一般性验证：

1. 图1：即使实际流水并行下bubble分布并不均匀，但可强制均匀编排，不影响性能

2. 图2：当forward和backward时钟周期不同的时，仍可按照理想状态编排，公式成立

3. 图3：num_micro_batches>pp_degree时同样可按理想状态编排，均匀分布bubble

GPU4				1	B1	2	B2	3	B3	4	B4				
GPU3			1	2		B1	3	B2	4	B3		B4			
GPU2		1	2	3			B1	4	B2		B3		B4		
GPU1	1	2	3	4				B1		B2		B3		B4	

1F1B下非理想状态流水并行图

GPU4				1	B1	2	B2	3	B3	4	B4							
GPU3			1		2	B1	3	B2	4	B3		B4						
GPU2		1	2			3	B1	4	B2			B3		B4				
GPU1	1	2	3				4	B1				B2		B3		B4		

1F1B下t_f!=t_b流水并行图

GPU4				1	B1	2	B2	3	B3	4	B4	5	B5					
GPU3			1		2	B1	3	B2	4	B3	5	B4		B5				
GPU2		1	2			3	B1	4	B2	5	B3			B4	B5			
GPU1	1	2	3				4	B1	5	B2				B3	B4	B5		

1F1B下num_micro_batches>pp_degree流水并行图

PART3：未来工作规划

>>>>>>>>

对于非均衡vpp编排的改进

- 降低非均衡vpp编排的bubble率，与均衡vpp的bubble率保持一致

对aa_diff检查的开发

- 当前研究了多种流水并行下的bubble编排策略，自动检查脚本还未开发
- 将aa_diff检查抽象封装，适配各种编排策略的检查调用

PART4：总结与体会

>>>>>>>>

护航计划成果总结

- 自动并行的非均衡vpp下，编排阶段支持acc_step % pp_degree != 0的情况
- 自动并行的非均衡vpp下，重切分阶段支持hidden_layer % chunk_nums != 0的情况
- 提出非均衡vpp编排的bubble优化策略
- 归纳总结aa_diff检查的bubble嵌入位置公式
- 做了一个40页的流水并行ppt串讲

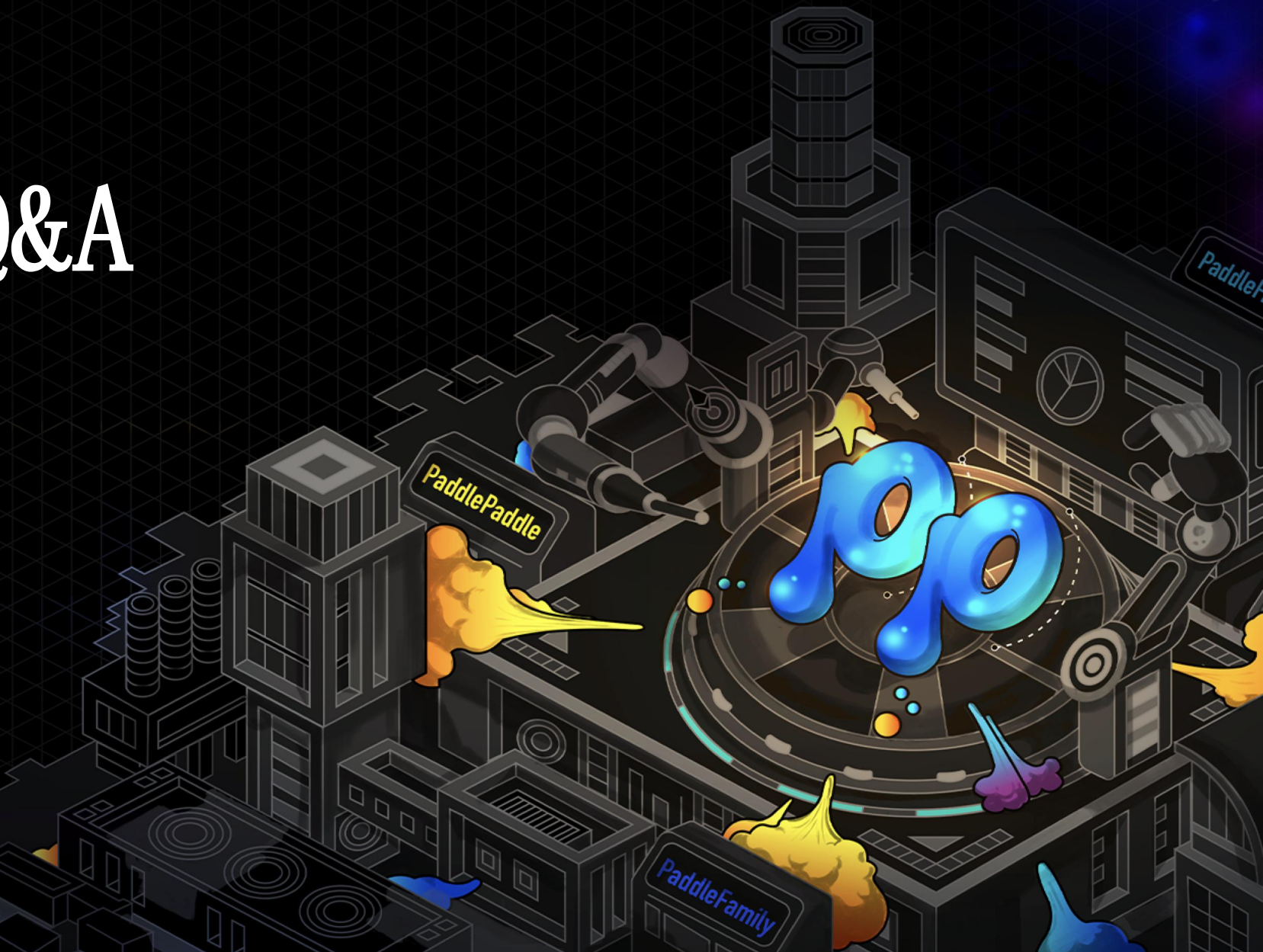
PART4：总结与体会

>>>>>>>>

个人体会

- 自动并行下的非均衡vpp适配工作是一个探索性的工作，没有现成的策略或实现以供参考
  - 现有资料通常只针对简单的情况，例如参数为整数倍关系
  - 现有论文一般只提供了常规关系下的vpp编排方式
- 非均衡vpp适配方法与aa_diff检查中bubble位置公式来自于大量绘制流水并行的时序图总结的经验
  - 自动并行即是将人的经验总结归纳并公式化，从而实现自动化的过程

Thanks! Q&A



答辩人：郑天宇 / zty-king
指导人：陈锐彪 / From00
飞桨护航计划集训营