深度学习技术及应用
国家工程研究中心 | 飞桨

静态图流水并行功能增强和性能优化

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RAG、计算机视觉、强化学习。第七期黑客松护航计划结束后，我对飞桨平台有了深入了解，同时对模型分布式训练—流水并行方向产生了更强烈的兴趣，于是我继续参加了第八期黑客松护航计划。本次参与护航计划，我主要在流水并行功能增强和性能优化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：自动并行下解决vpp策略通信死锁导致的hang问题

- vpp下拆解forward和backward，通过编排调度，解决生产环境下，vpp策略hang住的问题。

任务2：Locallayer API 优化

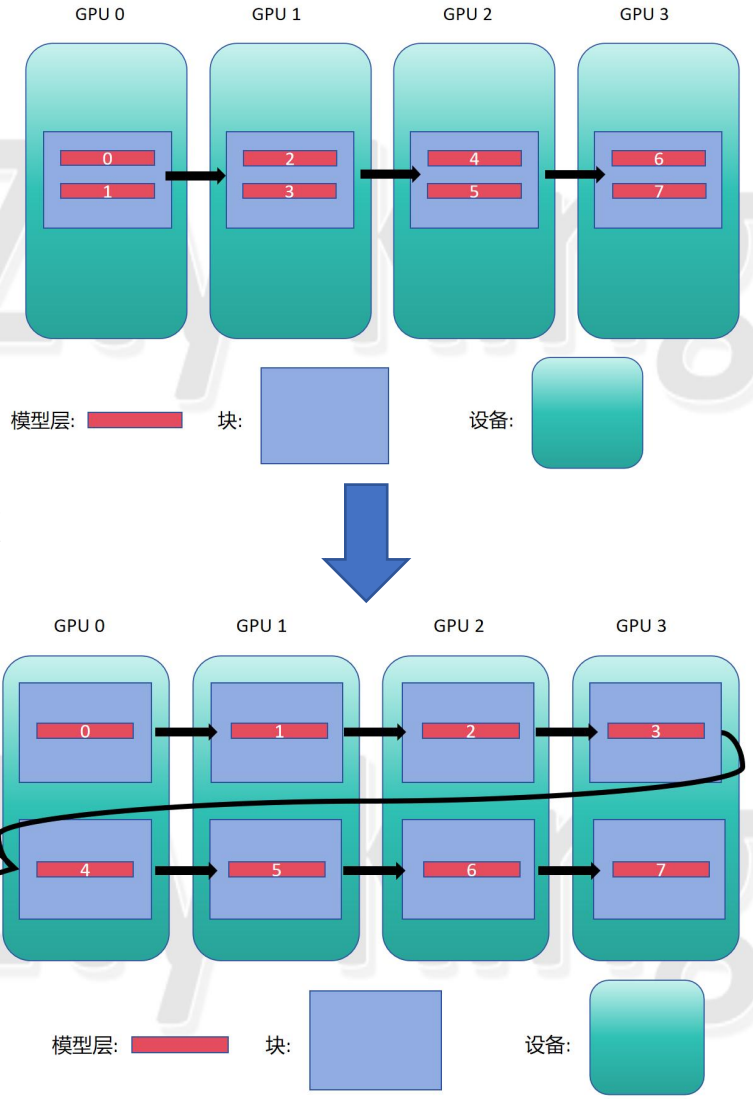
- 对标Locallayer，在此基础上优化，开发一个函数封装器local_map，函数经过封装后，允许用户将分布式张量传递给为普通张量编写的函数，使用户能够像编写单卡代码一样实现局部操作。
- 写了近10种单测示例验证动静半下local_map的正确性，同时替换Paddle，PaddleMIX和PaddleNLP中所有使用了LocalLayer的单测，并验证，同时修复一个存在潜在bug的单测

任务3：动态图的流水并行功能开发

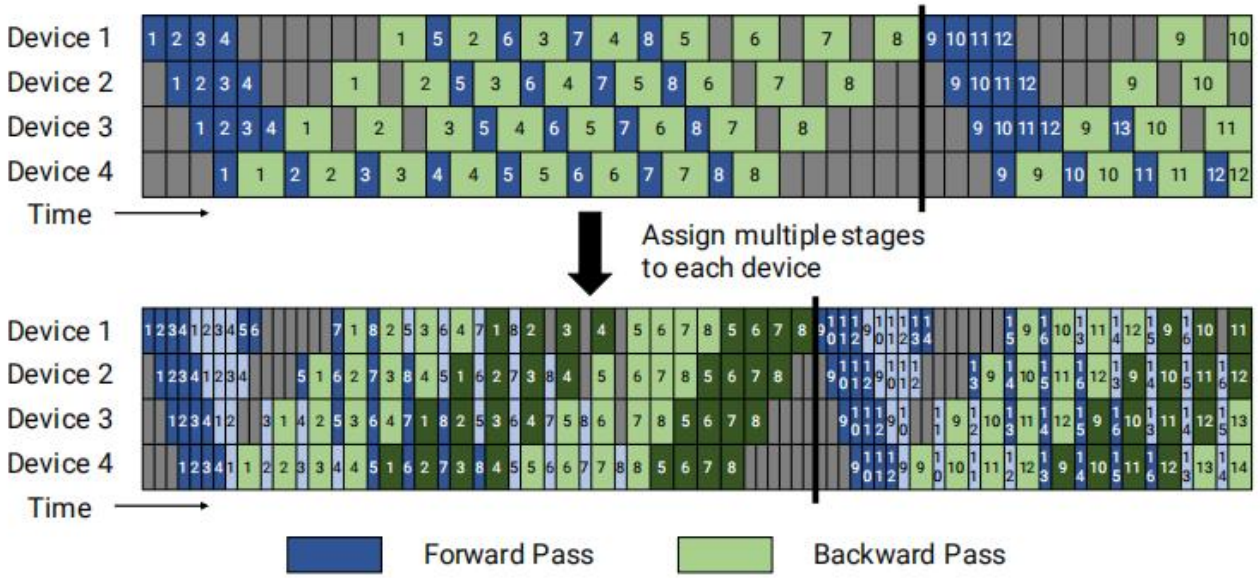
- 协助开发动半下pp相关的数十个相关组件，根据开发的基本组件完善单测，测试并修复潜在的bug

PART2：实际开发工作介绍-vpp流水并行(背景介绍)

重切分阶段



编排阶段



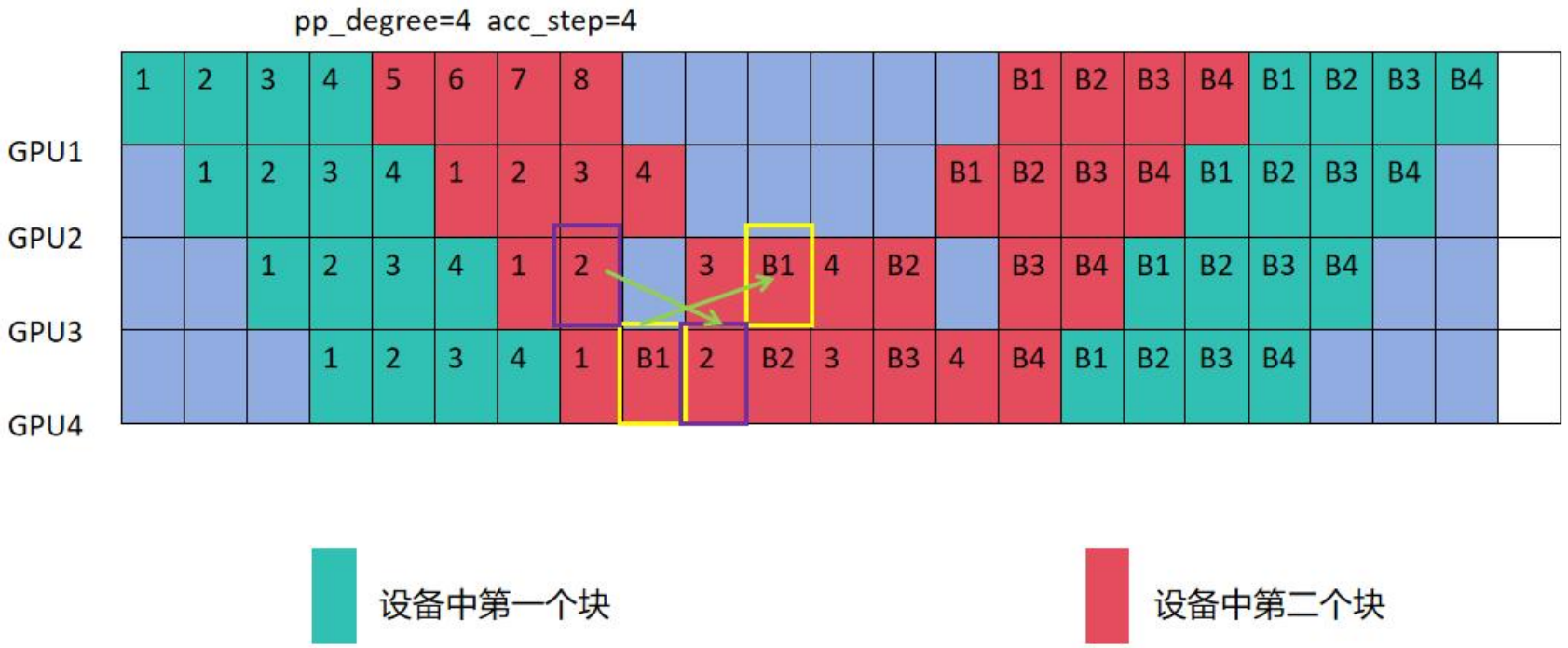
PART2：实际开发工作介绍-vpp下拆解forward和backward

>>>>>>>>

问题分析

micro_batch_2做forward后，send数据到GPU4
micro_batch_1做backward后，也先做send，讲数据发送到GPU3

此时，两个设备都在等待对方做recv操作接收数据，陷入通信死锁，导致hang



pp_degree=4,acc_step=8,vpp_degree=2

PART2：实际开发工作介绍-vpp下拆解forward和backward

>>>>>>>>>

方案设计

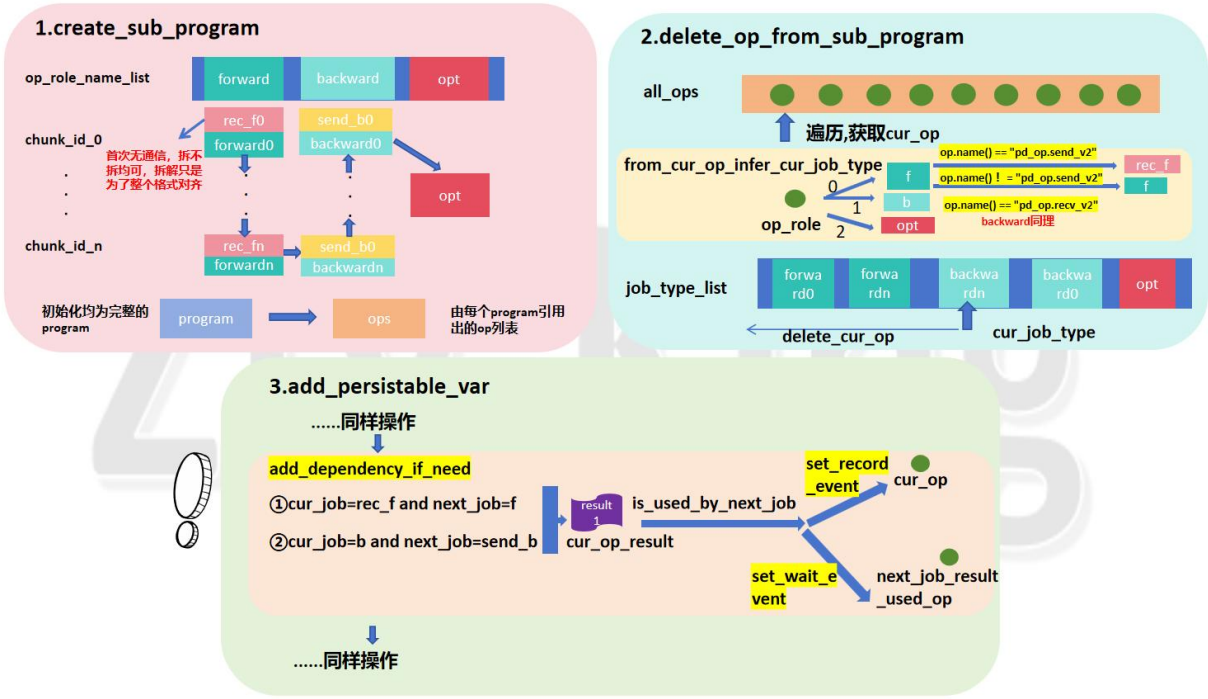
➢ 子图拆解：

1. 将forward拆解为通信模块recv_forward和计算模块forward

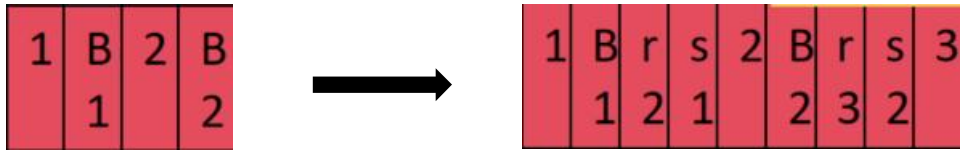
2. 将backward拆解为计算模块send_backward和通信模块send_backward。

➢ 编排改进：

1. 编排阶段让forward先接收，backward再发送



子图拆解



编排改进

PART2：实际开发工作介绍-vpp下拆解forward和backward

>>>>>>>>>

实现效果

- 编排变化：

1. steady阶段编排变化：

从1F1B变为BrsF

（r表示recv_forward）

(s表示send_backward)

2. 计算 warm_up_steps的变化 ：

在上面公式基础上新增一条公式：

```
warm_up_steps = min(total_num_steps, warm_up_steps + 1)
```

pp_degree=4,acc_step=4,vpp_degree=2

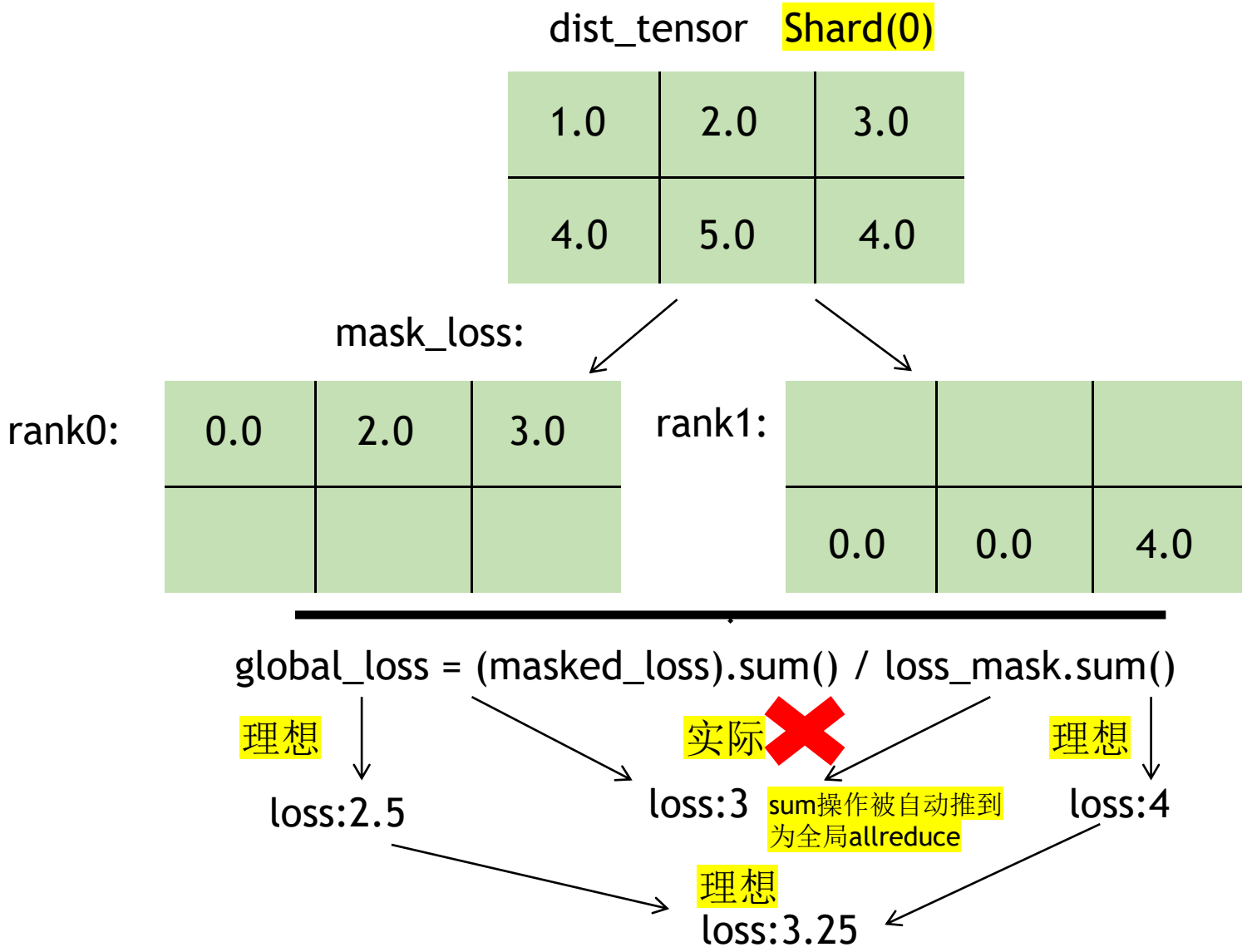
拆解后，先接收再发送，不再陷入hang

PART2：实际开发工作介绍-LocalLayer API优化

>>>>>>>>>

背景介绍

- 在动半、静半分布式训练中,经常需要将分布式张量(dist_tensor)传递给为仅仅能处理普通张量(dense_tensor)或者必须以本地视角处理本地张量的函数。



PART2：实际开发工作介绍-LocalLayer API优化

>>>>>>>>

问题分析

当前开发的Locallayer对于Layer含参数的情况，很难处理

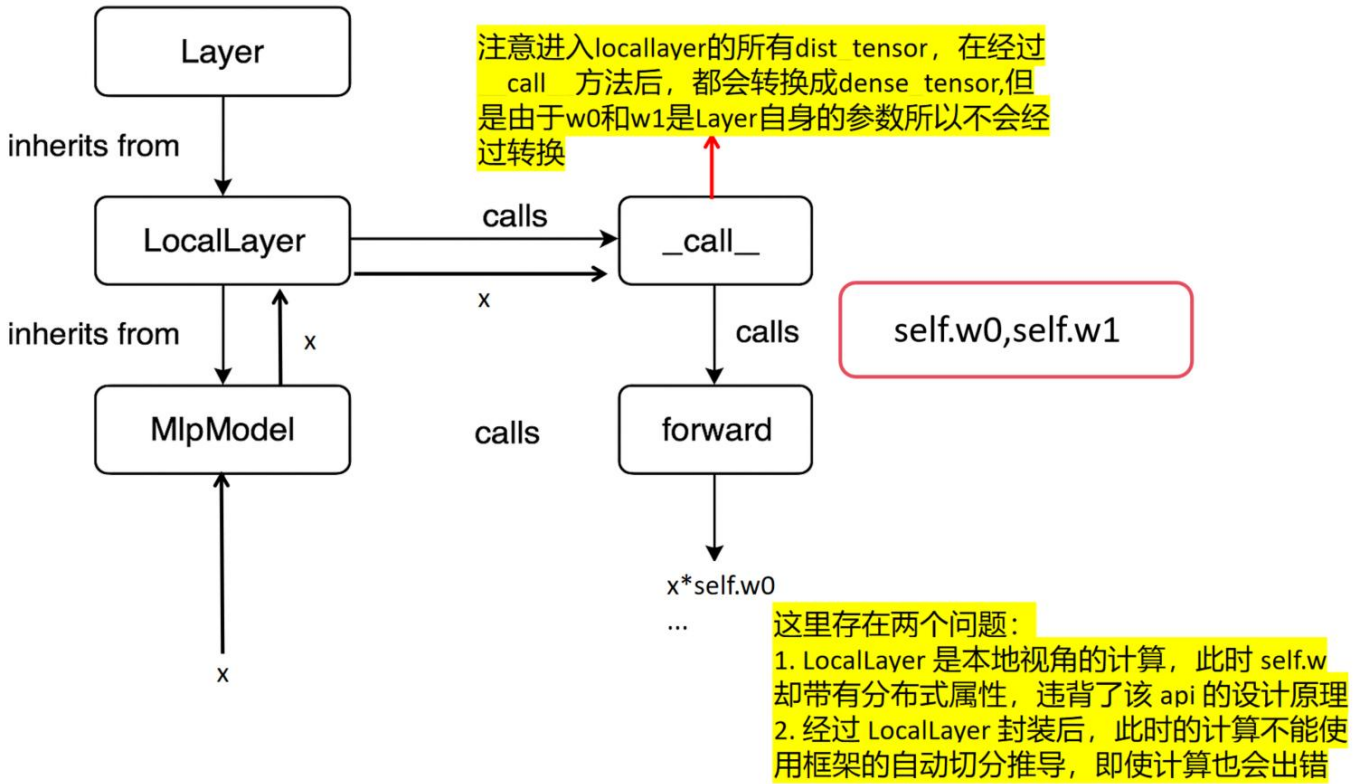
```python
# 创建 1D 的 ProcessMesh，4 个设备
mesh = dist.ProcessMesh([0, 1, 2, 3], dim_names=['x'])

class MlpModel(dist.LocalLayer):
    def __init__(self):
        super(MlpModel, self).__init__()

        # w0 按列切分：Shard(1)
        self.w0 = dist.shard_tensor(
            self.create_parameter(shape=[1024, 4096]),
            mesh,
            [dist.Shard(1)]
        )

        # w1 按行切分：Shard(0)
        self.w1 = dist.shard_tensor(
            self.create_parameter(shape=[4096, 1024]),
            mesh,
            [dist.Shard(0)]
        )

    def forward(self, x):
        x = paddle.matmul(x, self.w0)
        z = paddle.matmul(x, self.w1)
        return z
```



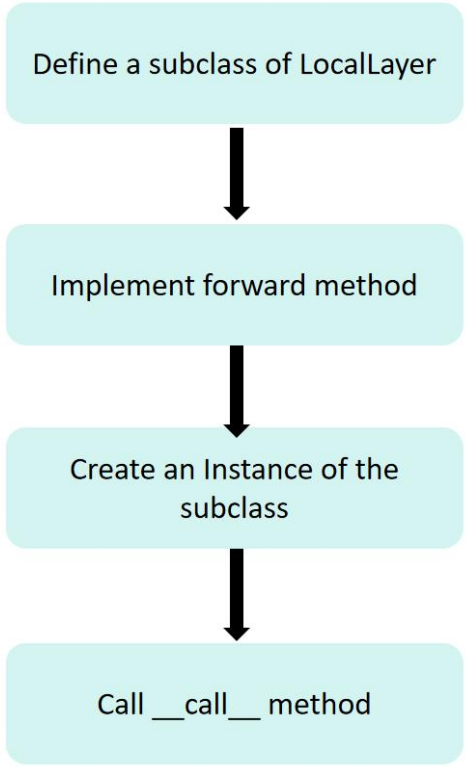
左侧示例的部分计算流程图

PART2：实际开发工作介绍-LocalLayer API优化

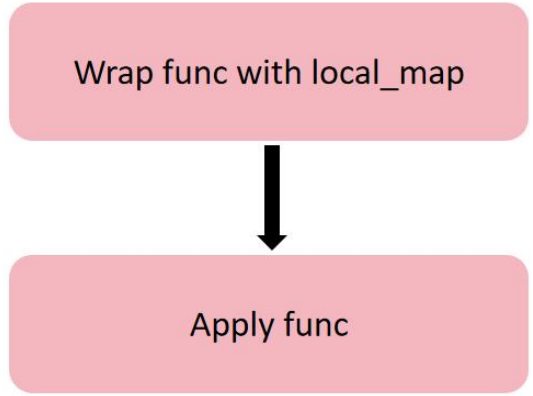
>>>>>>>>>

方案设计

- 以函数形式开发，而不再是Layer结构。
- 添加reshard参数，由用户自由决定是否需要对输入的dtensor做reshard。
- 更改API命名为local_map与PyTorch对齐，便于用户迁移



Locallayer 接口形式



local_map 接口形式

PART2：实际开发工作介绍-LocalLayer API优化

>>>>>>>>>

实现效果

方面	LocalLayer	local_map
封装方式	类封装，继承 Layer，重写 forward	函数式包裹，接入即用
语法负担	必须定义类 + 结构复杂	一行代码即可创建分布式兼容函数
数据处理入口	__call__ 自动改写输入输出，可能会误用自身的参数	显式传参，灵活组合
状态管理	需要管理 Layer 状态	不存在状态概念
reshard支持	❌ 无法做 reshard	✅ 可以根据用户需求，批量对输入的 dtensor 做 reshard
混合输出支持	❌ 只能输出 dtensor	✅ 可以输出 dtensor、tensor，以及一些非 tensor 的数值

从“写个类、注册属性、重写forward”变成“包个函数、直接用”——local_map 让并行封装像写普通函数一样简单。

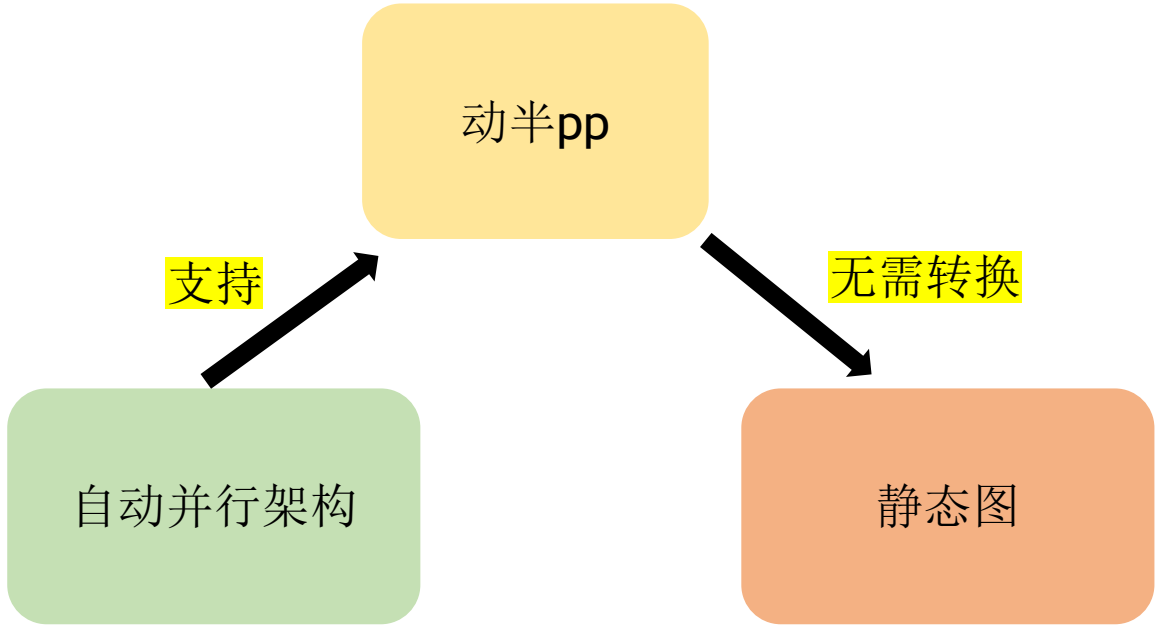
PART2：实际开发工作介绍-动态图的流水并行功能开发

>>>>>>>>>

背景介绍

1. 当前仅在动手和静半两个场景下支持pp，动半下没有pp相关组件，不支持pp。

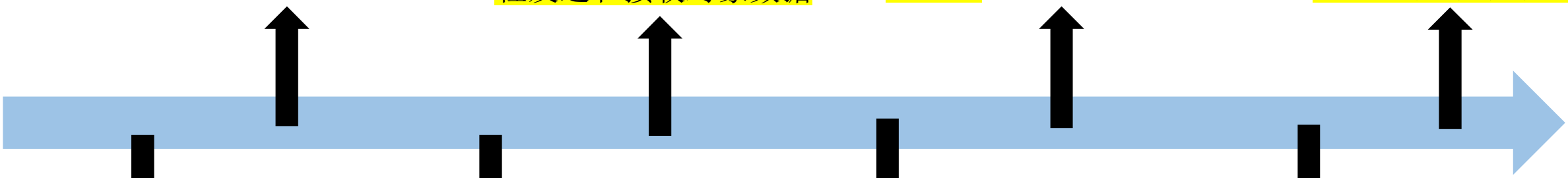
2. 动手无法使用自动并行架构的一套自动推导逻辑。



PART2：实际开发工作介绍-动态图的流水并行功能开发

>>>>>>>>>

动半PP开发工作概览



Lazy Initialization
添加单测

send_object_list
recv_object_list
协助开发
在shape_inference过程发送和接收对象数据

GET_PP_INFO_OF_LAYER
独立开发
让用户在使用pp策略时，正确切分模型到对应的mesh上

bug_fix：get_group
每次get_group都会调用new_group创建新的通信组，会导致重复创建通信组，会导致显存爆炸等错误

get_submesh_with_dim
get_group
添加英文文档与单测

microbatch组件：
_split_args_helper
merge_chunks等
协助开发
切分输入数据得到micro_batch

PipelineStage
协助开发
包含构建流水并行过程一个stage的每个阶段的组件

Schedules
协助开发
定义PipelineScheduleSingle和PipelineScheduleMulti并基于此实现FthenB、1F1B、VPP

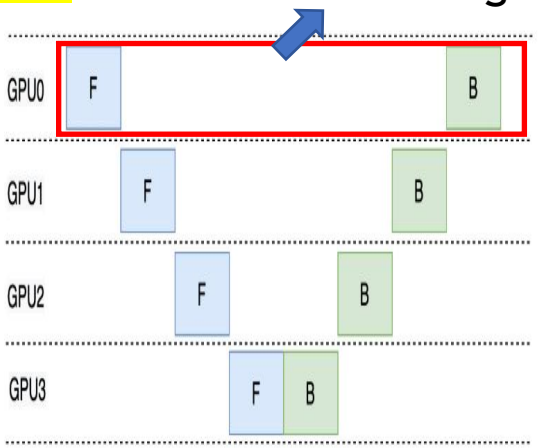
PART2：实际开发工作介绍-动态图的流水并行功能开发

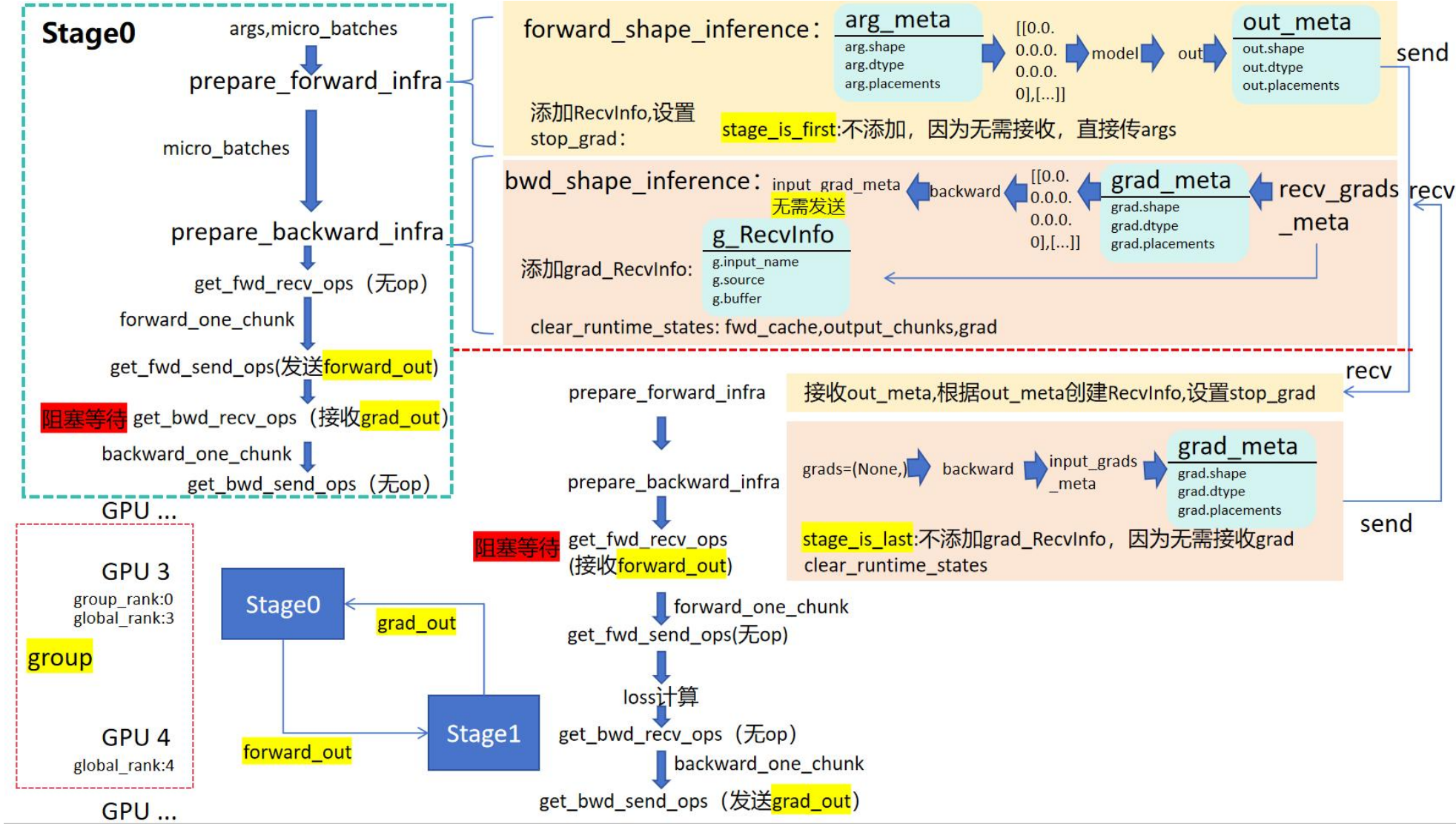
>>>>>>>>>

PipelineStage设计：

用来构建流水并行过程一个stage的每个阶段。

开发目的：Schedule是基于PipelineStage设计多种pp策略的





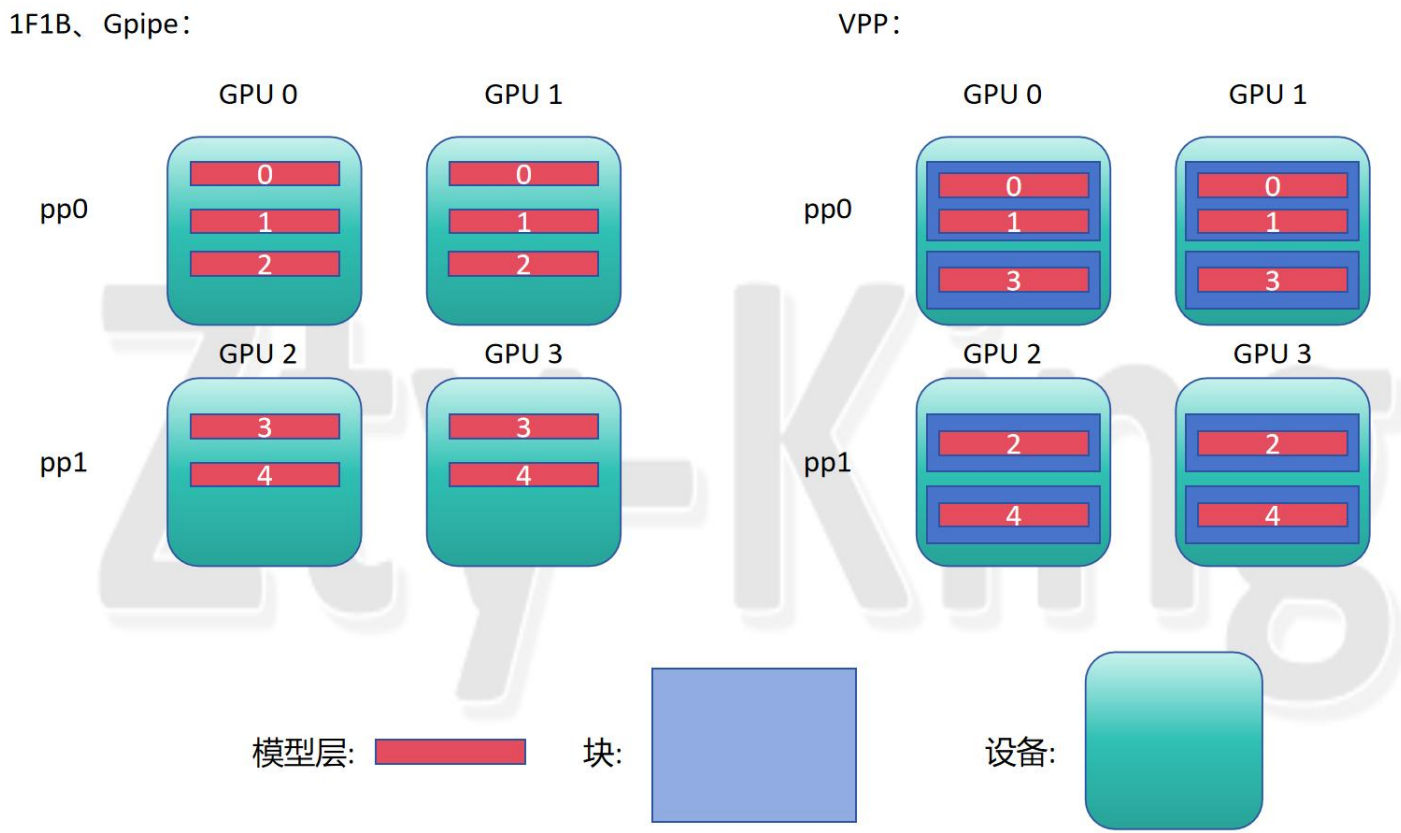
PART2：实际开发工作介绍-动态图的流水并行功能开发

>>>>>>>>>

GET_PP_INFO_OF_LAYER组件：

为了让用户在使用pp策略时，正确切分模型到对应的mesh上，开发此工具。

将根据 hidden_layer_num, mesh, pp_schedule, vpp_degree，来处理模型切分，并根据layer_index,返回对应的mesh信息，也可一次返回全部layer的mesh信息。



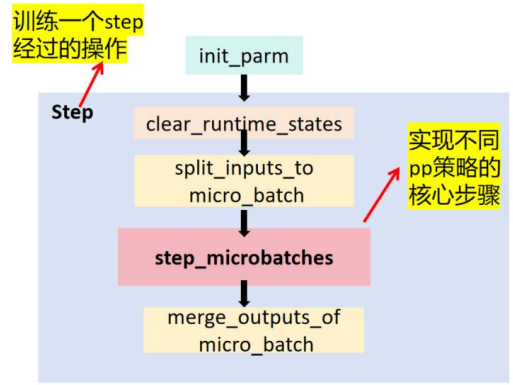
PART2：实际开发工作介绍-动态图的流水并行功能开发

>>>>>>>>

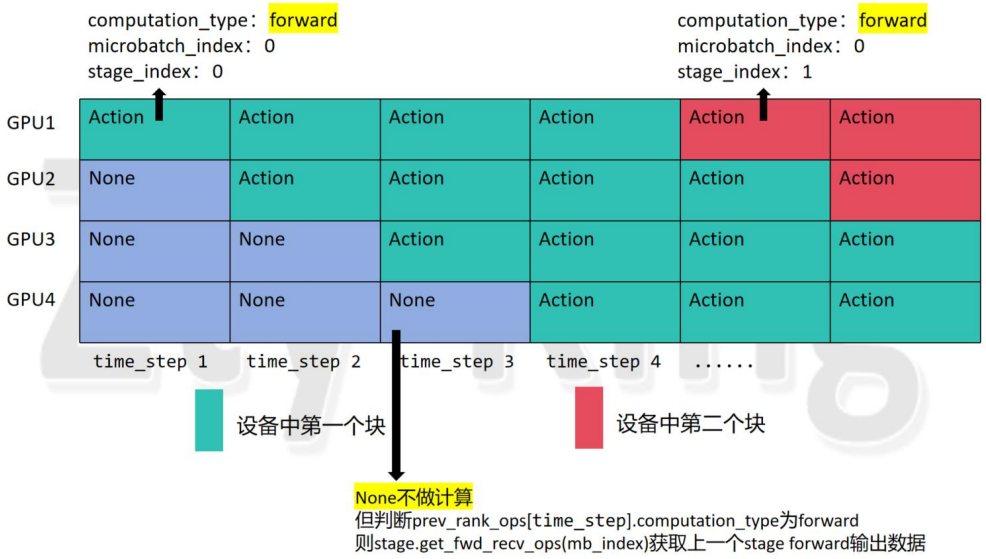
Schedules设计思路：

1.对于每个rank只分配一个stage的，我们开发组件PipelineScheduleSingle，每个继承该类的pp策略重写step_microbatches方法

2.PipelineScheduleMulti则实现了step_microbatches，继承该类的pp策略无需再写该方法。但需要初始化self.pipeline_order，即每个rank执行的_Action列表，_Action.computation_type包含forward、backward、None等操作(None表示当前时间步不做操作，即bubble)



PipelineScheduleSingle流程



PipelineScheduleMulti的step_microbatches核心流程

PART2：实际开发工作介绍-动态图的流水并行功能开发

>>>>>>>> Splitting Fronted(用户对模型做切分标记)

Schedules实现效果：

1. 该组件主要开发了三种pp策略：

single_stage:Gpipe、1F1B

multi_stage:VPP

2. 测试了三种策略；与同样数据和参数的动半朴素流水并行对齐了精度(数据采用float32,对齐7位有效数字)；同时测试了dp、pp混合的case。

```python
class PPMyModel_SingleStage(nn.Layer):
    def __init__(self):
        super().__init__()
        self.mesh=paddle.distributed.ProcessMesh([0,1,2,3], dim_names=["pp"])
        self.num_layers = 8
        self.num_layers_per_card = 2
        self.linears = nn.LayerList()
        for i in range(self.num_layers):
            linear = nn.Linear(8, 8, bias_attr=False)

            linear.weight = dist.shard_tensor(
                linear.weight,
                self.get_pp_mesh(i),
                [dist.Replicate()],
            )

            self.linears.append(linear)

    def get_pp_mesh(self, layer_index):
        # layer_index=0-7 对应的 mesh_idx 分别为 0,0,1,1,2,2,3,3
        mesh_idx = int(layer_index // self.num_layers_per_card)
        return self.mesh[mesh_idx]

    def forward(self, x):
        x.stop_gradient = False
        out = x
        device_id = dist.get_rank()
        for i in range(self.num_layers):
            if int(i // self.num_layers_per_card) == device_id:
                out = self.linears[i](out)

        return paddle.cast(out, 'float32')
```

将layer给shard到不同的mesh上

不同rank执行其对应layer的forward

注意这里无需将out，shard到下一个pp_stage,封装的组件会自动执行p2p通信

PipelineScheduleSingle

```python
class PPMyModel_MultiStage(nn.Layer):
    def __init__(self):
        super().__init__()
        self.mesh=paddle.distributed.ProcessMesh([0,1,2,3], dim_names=["pp"])
        self.num_layers = 8
        self.linears = nn.LayerList()
        for i in range(self.num_layers):
            linear = nn.Linear(8, 8, bias_attr=False)

            linear.weight = dist.shard_tensor(
                linear.weight,
                self.get_pp_mesh(i),
                [dist.Replicate()],
            )

            self.linears.append(linear)

    def get_pp_mesh(self, layer_index):
        mesh_idx = int(layer_index % 4)
        return self.mesh[mesh_idx]

    def forward(self, x):
        # 对于MultiStage，我们将模型层分片，因此forward调用 _Pipeline_model_chunk.forward
        pass

class _Pipeline_model_chunk(nn.Layer):
    def __init__(self, layers):
        super(_Pipeline_model_chunk, self).__init__()
        self.layers = layers
    def forward(self, x):
        out = x
        for layer in self.layers:
            out = layer(out)
        return out

def test_ScheduleInterleaved1F1B(self):
    fix_seeds()
    self.model = PPMyModel_MultiStage()
    self.local_stages=2
    self.micro_batches = 8
    self.stage_list=[]
    for i in range(self.local_stages):
        stage_model = _Pipeline_model_chunk(self.model.linears[self.rank+i*4:self.rank+i*4+1])
        self.stage_list.append(PipelineStage(stage_model,self.rank+i*4,8,group=self.group))
        self.stage_list[i].has_backward = True
```

vpp同样shard layer，但注意layer分配方式不同

这里由于同一rank上的不同layer要处理不同的数据(因为是不同的stage)，因此用pipeline_model_chunk将不同chunk的层独立出来）

PipelineScheduleMulti

PART3：未来工作规划

>>>>>>>>

动态图的流水并行功能开发

- 动半pp的save-load场景存在精度不齐的问题，待修复
- 自动化Splitting Fronted，用户只需输入少量参数即可自动分割模型到不同设备上运行。(待定)
- Schedules组件的持续优化——如vpp适配更多非均衡情况、替换掉手动编排None的操作
- 继续开发动半pp的相关组件

PART4：总结与体会

>>>>>>>>

护航计划成果总结

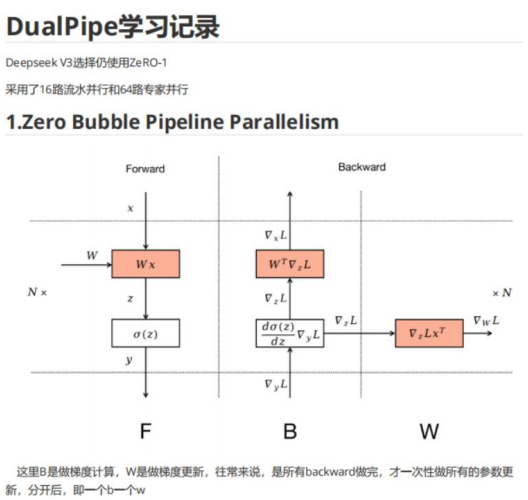
- 自动并行拆分vpp编排的forward和backward中的通信部分，修复生产下hang住的问题。
- 优化Locallayer，对标开发了local_map API。
- 协同开发动半pp相关的数十个组件，并编写大量单测

测试，理解组件设计逻辑，从而发现组件代码中的问题，

并进行bug修复

- 自我归纳总结出了dualpipe编排规律，并开发其编排代码。
- 记录20多页的dualpipe策略分析，以及10多页

的locallayer动半，动手执行的日志分析。



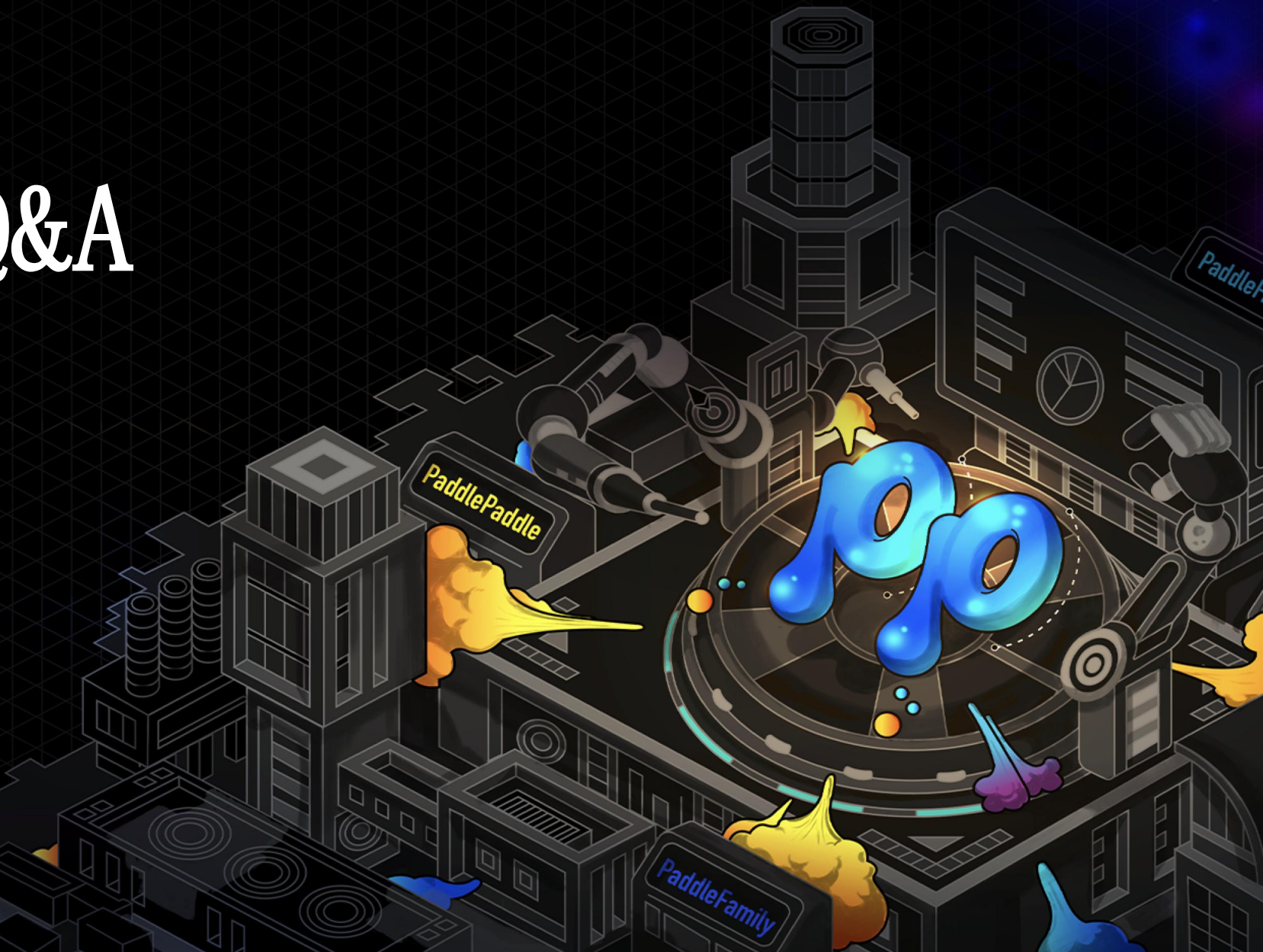
PART4：总结与体会

>>>>>>>>

个人体会

- 流水并行策略的开发无法完全考虑到生产出现的每一个问题，但是要符合规范，并且要能学会从问题中发现解决办法。
- API的设计与开发要足够合理化，不仅仅要以设计者的角度思考问题，更要考虑到用户实际使用过程中可能会出现的潜在问题，并确保API接口对客户的使用体验友好。

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