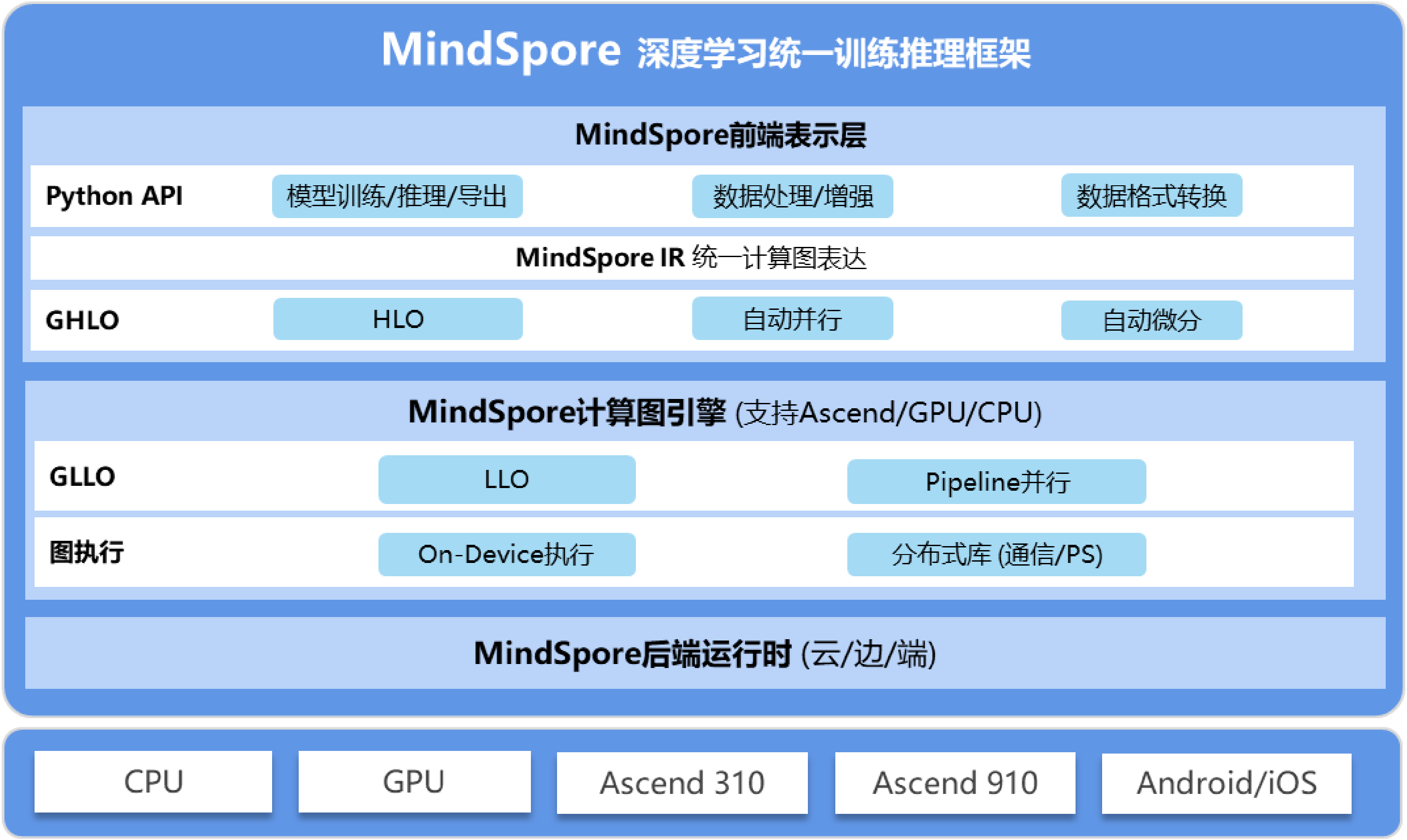
MindSpore介绍
——分布式自动并行训练

龚子妍

MindSpore

MindSpore是什么



目录

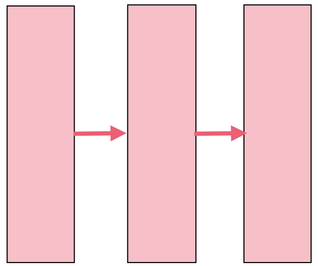
contents

MindSpore

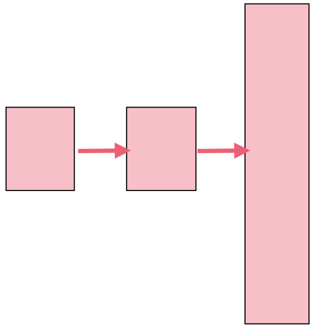
神经网络训练趋势

MindSpore

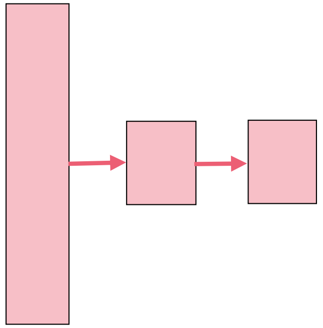
NLP：Transformer

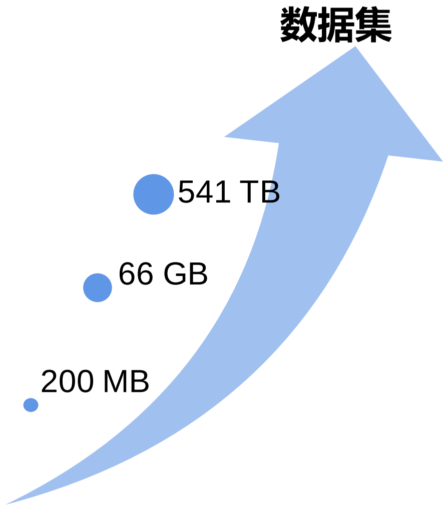


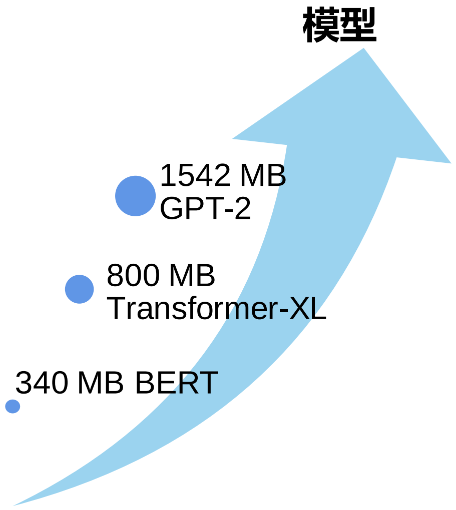
人脸识别：ReID



推荐：Wide&Deep

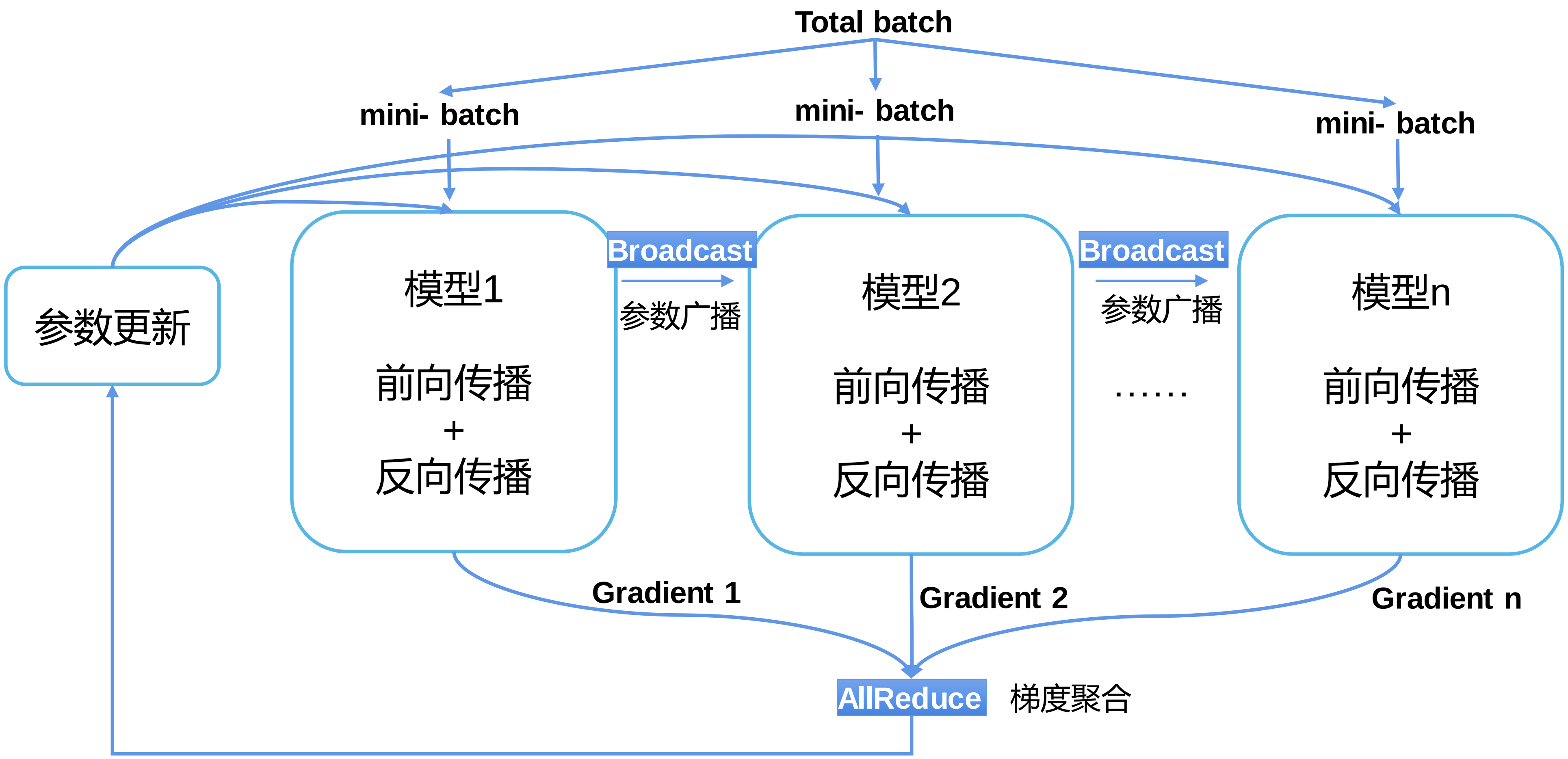






单卡执行，耗时长，内存小　　利用更多的机器！

数据并行图解

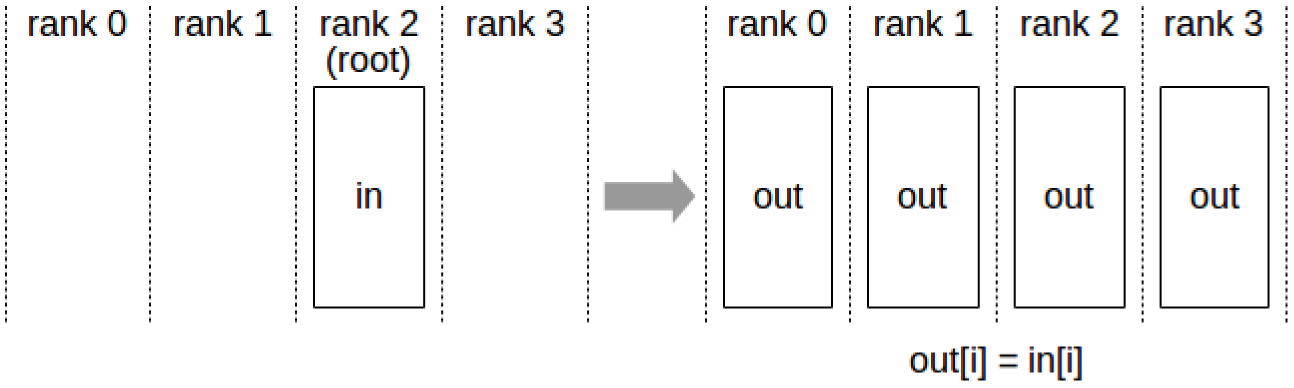


每卡跑同样的模型，处理不同的样本数据

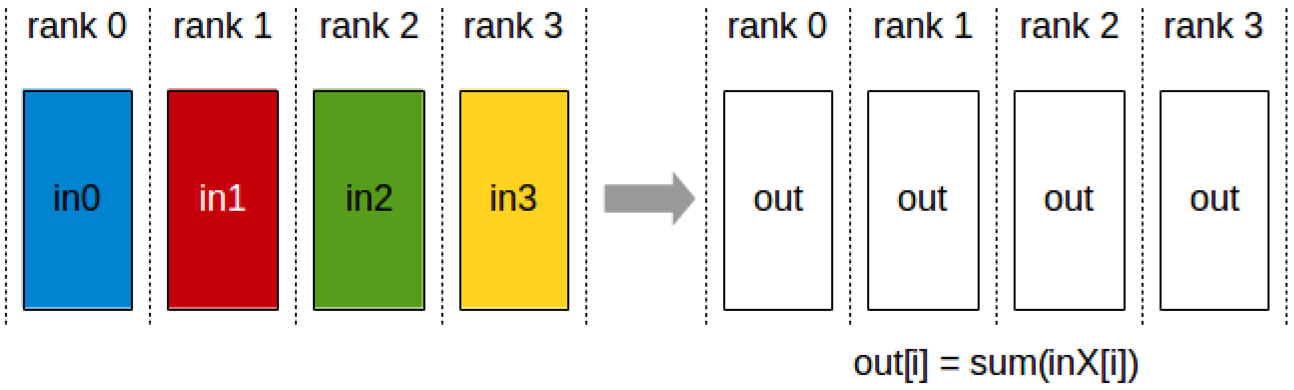
集合通信

MindSpore

Broadcast

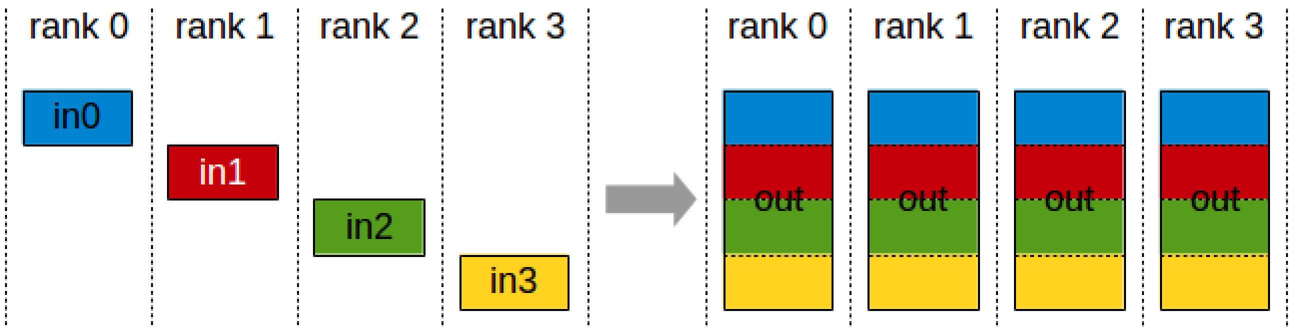


AllReduce



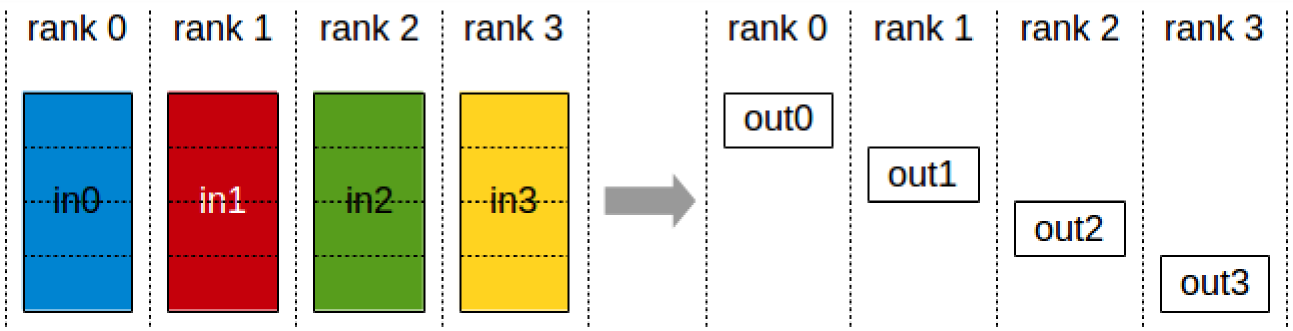
集合通信

AllGather



out[Y*count+i] = inY[i]

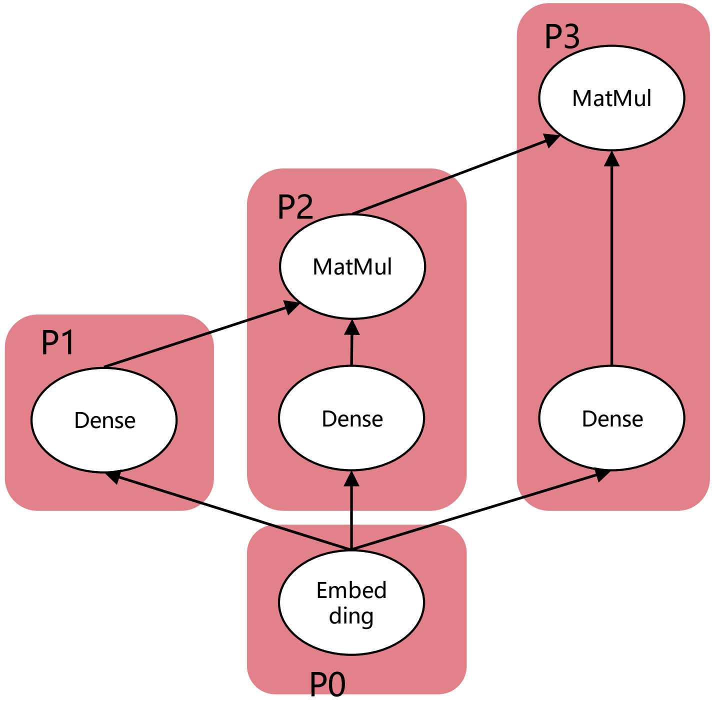
ReduceScatter



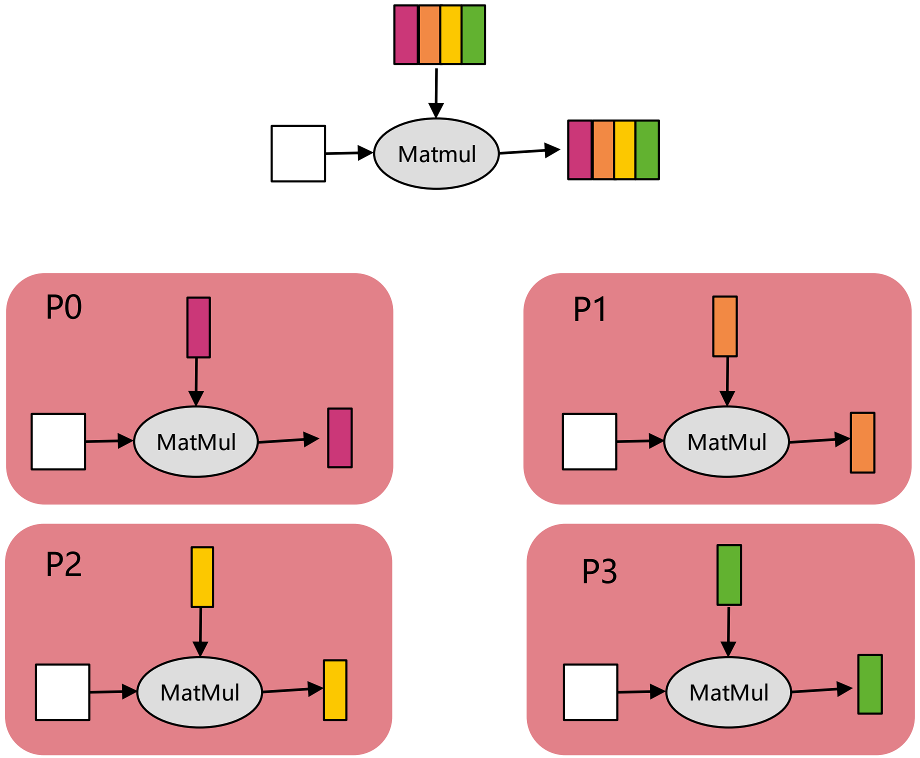
outY[i] = sum(inX[Y*count+i])

模型并行图解

层间模型并行：模型以层为单位切分到多个设备

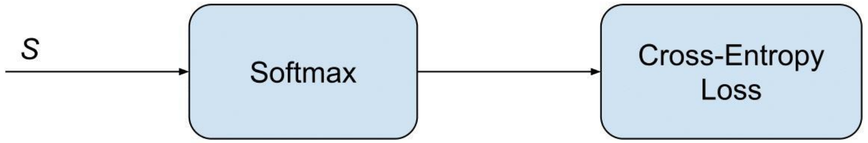


层内模型并行：每层的模型参数切分到多个设备



手动切分的难度

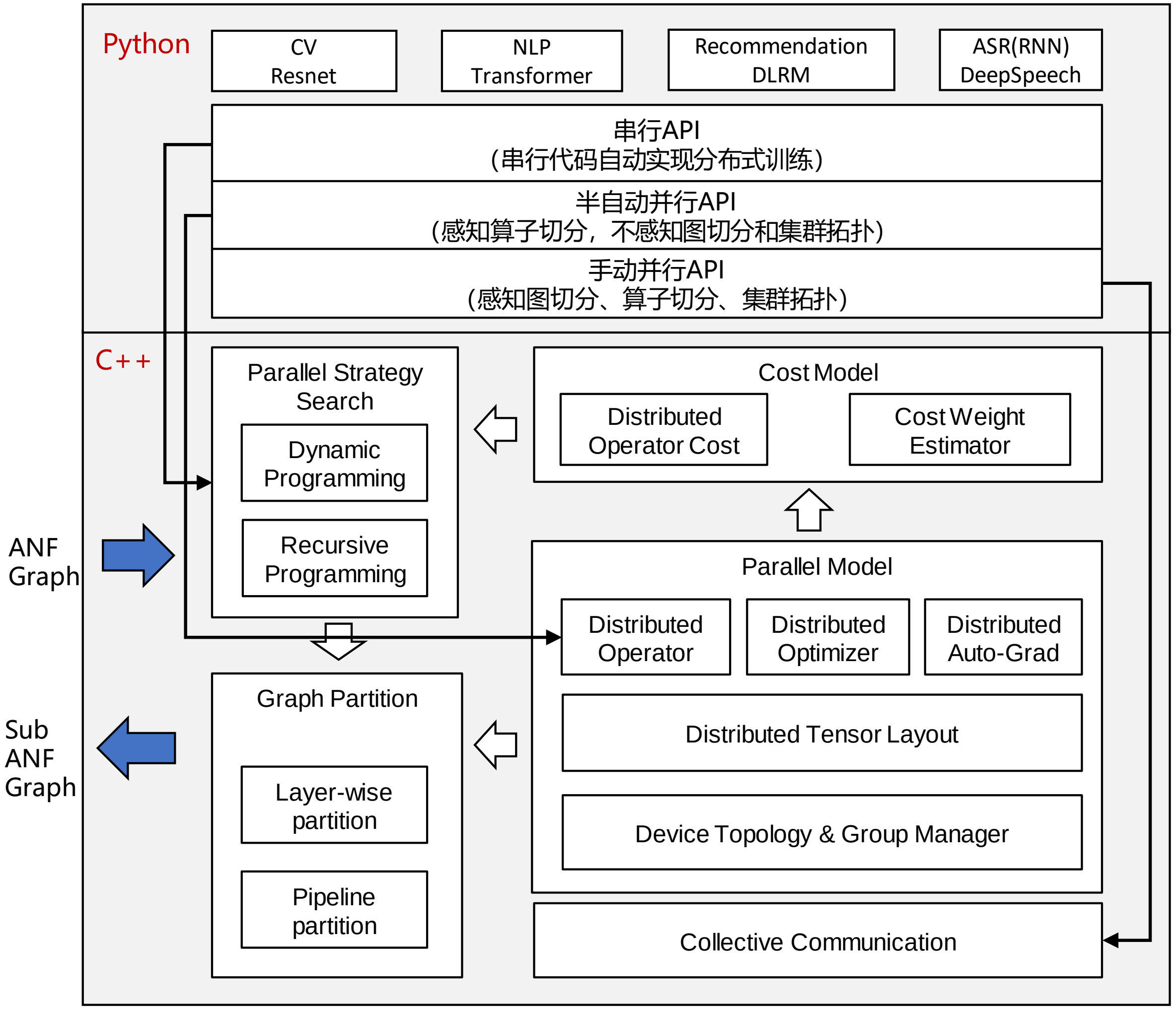
内存上限、通信代价、张量排布



$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}} \qquad CE = -\sum_i^C t_i log(f(s)_i)$$

```python
def forward(total_features_r, metric_fc_weights, local_label):
    norm_total_features = F.normalize(total_features_r)
    w = F.normalize(metric_fc_weights)
    local_logit = F.linear(norm_total_features, w)

    # SoftmaxCrossEntropy
    local_logit_max, _ = torch.max(local_logit, dim=-1, keepdim=True)
    # get global logit max
    dist.all_reduce(local_logit_max, op=dist.ReduceOp.MAX)

    local_exp = torch.exp(local_logit)
    local_exp_sum = torch.sum(local_exp, dim=-1)
    # get global exp sum
    dist.all_reduce(local_exp_sum, op=dist.ReduceOp.SUM)

    global_exp_sum = local_exp_sum[:, None]
    local_softmax_result = local_exp / global_exp_sum

    local_loss = torch.sum((-1 * torch.log(local_softmax_result+eps) * local_label), dim=-1)
    # get global loss
    dist.all_reduce(local_loss, op=dist.ReduceOp.SUM)
    local_loss = torch.mean(local_loss)
```

MindSpore自动并行方案

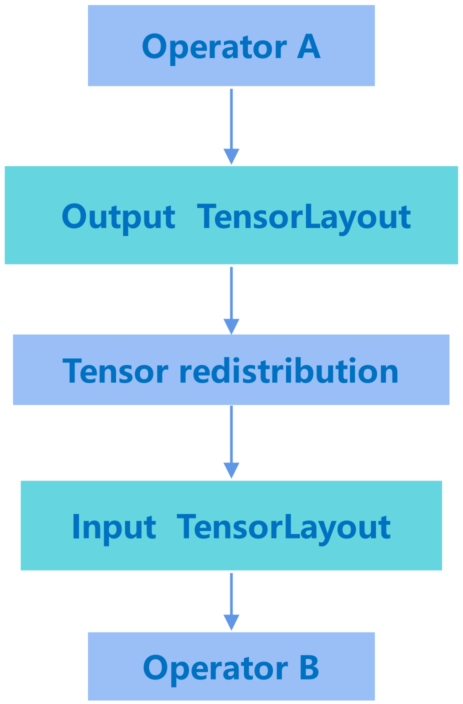


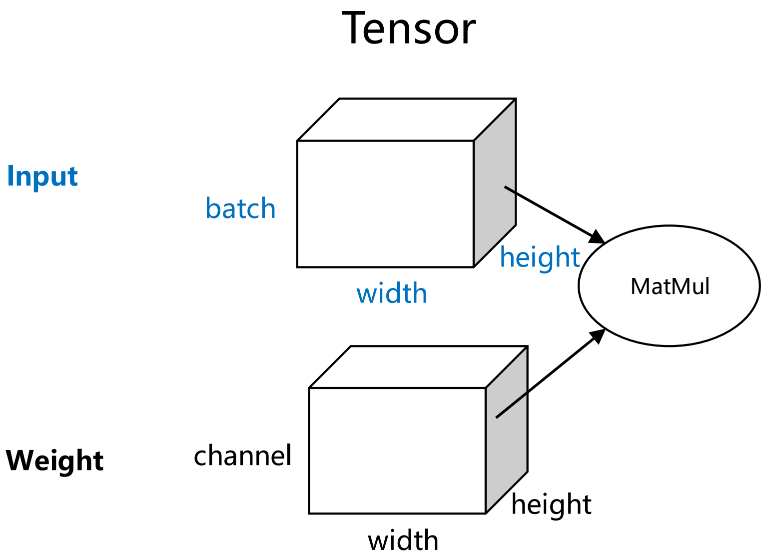
打破样本和参数的边界，按计算数据维度切分，实现混合并行。

1. 算子自动切分
2. 整图自动切分
3. 集群拓扑感知调度
4. 最优切分策略自动搜索

算子切分定义

亮点1：通用的算子切分表达和张量排布模型





并行切分约束：
各维度数据、计算和资源需要均匀切分，并以2为基。

并行维度

Input：
d0: tensor.batch
d1: tensor.height
d2: tensor.width

Weight：
d0: tensor.channel
d1: tensor.height
d2: tensor.width

切分策略

set_strategy(([d0, d1, d2], [d2, d1, d0]))
数据并行
set_strategy(([dev_num, 1, 1], [1, 1, 1]))
模型并行
set_strategy(([1, 1, 1], [1, 1, dev_num]))

数据并行转模型并行样例

典型场景：ReID

$$Z=(X\times W)\times V$$

1. $Y=\begin{bmatrix}X_1\\X_2\\X_3\\X_4\end{bmatrix}\times W=\begin{bmatrix}Y_1\\Y_2\\Y_3\\Y_4\end{bmatrix}$

2. $Z=\mathrm{Y}\times[\,V_1\quad V_2\quad V_3\quad V_4]=[\,Z_1\quad Z_2\quad Z_3\quad Z_4]$

```python
class DenseMatMulNet(nn.Cell):
    def __init__(self):
        super(DenseMutMulNet, self).__init__()
        self.matmul1 = ops.MatMul.set_strategy({[4, 1], [1, 1]})
        self.matmul2 = ops.MatMul.set_strategy({[1, 1], [1, 4]})
    def construct(self, x, w, v):
        y = self.matmul1(x, w)
        z = self.matmul2(y, v)
        return s
```

	X按行切分　数据并行		V按列切分　模型并行
D1	$X_1 \rightarrow W \rightarrow X_1W_1 = Y_1$	Tensor Redistribution (allgather) ⇨	$\mathrm{Y} \rightarrow V_1 \rightarrow YV_1 = Z_1$
D2	$X_2 \rightarrow W \rightarrow X_2W_2 = Y_2$		$\mathrm{Y} \rightarrow V_2 \rightarrow YV_2 = Z_2$
D3	$X_3 \rightarrow W \rightarrow X_1W_3 = Y_3$		$\mathrm{Y} \rightarrow V_3 \rightarrow YV_3 = Z_3$
D4	$X_4 \rightarrow W \rightarrow X_2W_4 = Y_4$		$\mathrm{Y} \rightarrow V_4 \rightarrow YV_4 = Z_4$
	Output tensor layout		**Input tensor layout**

MindSpore

模型并行转数据并行样例

典型场景：DLRM

$$Z = (X \times W) \times V$$

1. $Y = [X] \times [W_1 \quad W_2 \quad W_3 \quad W_4] = [Y_{c1} \quad Y_{c2} \quad Y_{c3} \quad Y_{c4}]$

2. $Z = \begin{bmatrix} Y_{b1} \\ Y_{b2} \\ Y_{b3} \\ Y_{b4} \end{bmatrix} \times [\,V\,] = \begin{bmatrix} Z_1 \\ Z_2 \\ Z_3 \\ Z_4 \end{bmatrix}$

```
class DenseMatMulNet(nn.Cell):
    def __init__(self):
        super(DenseMutMulNet, self).__init__()
        self.matmul1 = ops.MatMul.set_strategy({[1, 1], [1, 4]})
        self.matmul2 = ops.MatMul.set_strategy({[4, 1], [1, 1]})
    def construct(self, x, w, v):
        y = self.matmul1(x, w)
        z = self.matmul2(y, v)
        return s
```

	Channel Dimension					Tensor Redistribution (alltoall)	Batch Dimension				
D1	X →	W_{c1} →	$X_1 \times W_{c1}$	=	Y_{c1}	⇒	Y_{b1} →	V →	$H_{b1}W$	=	Z_{b1}
D2	X →	W_{c2} →	$X_2 \times W_{c2}$	=	Y_{c2}		Y_{b2} →	V →	$H_{b2}W$	=	Z_{b2}
D3	X →	W_{c3} →	$X_1 \times W_{c3}$	=	Y_{c3}		Y_{b3} →	V →	$H_{b3}W$	=	Z_{b3}
D4	X →	W_{c4} →	$X_2 \times W_{c4}$	=	Y_{c4}		Y_{b4} →	V →	$H_{b4}W$	=	Z_{b4}

数据并行叠加模型并行样例

典型场景：Transformer

$$Z = (X \times W) \times V$$

1. $Y = X \times W = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \times [\, W_1 \quad W_2 \,] = \begin{bmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{bmatrix}$

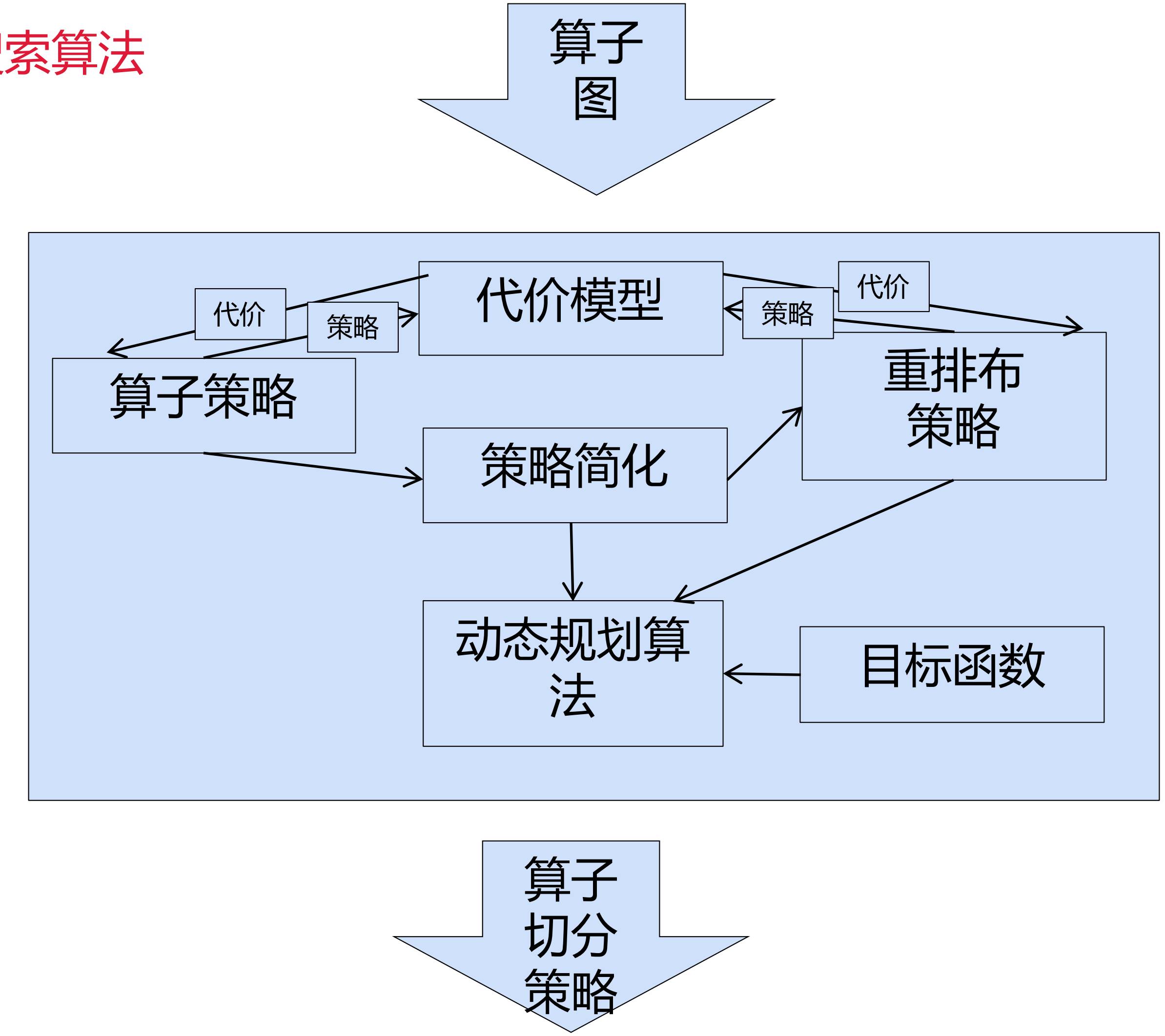
2. $Z = Y \times V = \begin{bmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{bmatrix} \times \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix}$

```
class DenseMatMulNet(nn.Cell):
    def __init__(self):
        super(DenseMutMulNet, self).__init__()
        self.matmul1 = ops.MatMul.set_strategy({[2, 1], [1, 2]})
        self.matmul2 = ops.MatMul.set_strategy({[2, 2], [2, 1]})
    def construct(self, x, w, v):
        y = self.matmul1(x, w)
        z = self.matmul2(y, v)
        return s
```

D1	$X_1 \longrightarrow W_1 \longrightarrow X_1W_1$	=	$Y_{11} \longrightarrow V_1 \longrightarrow Y_{11}V_1$	Allreduce	Z_1
D2	$X_1 \longrightarrow W_2 \longrightarrow X_2W_2$	=	$Y_{12} \longrightarrow V_2 \longrightarrow Y_{12}V_2$		Z_1
D3	$X_2 \longrightarrow W_1 \longrightarrow X_1W_1$	=	$Y_{21} \longrightarrow V_1 \longrightarrow Y_{21}V_1$	Allreduce	Z_2
D4	$X_2 \longrightarrow W_2 \longrightarrow X_2W_1$	=	$Y_{21} \longrightarrow V_2 \longrightarrow Y_{22}V_2$		Z_2

策略搜索原理

亮点2：高效的并行策略搜索算法



分布式自动微分

MindSpore

亮点3：便捷的分布式自动微分

利用自动微分流程，实现分布式反向的自动生成，避免复杂的手动微分过程。

```
def forward(ctx, x, metric_fc_weights):
        ...
        grad_of_weights = autograd.grad(local_logit, metric_fc_weights, retain_graph=True,
                                        grad_outputs=grad_of_logit)[0]/per_batch_size
        grad_of_x = autograd.grad(local_logit, total_features_r, grad_outputs=grad_of_logit)[0]/per_batch_size

        dist.all_reduce(grad_of_x, op=dist.ReduceOp.SUM)

        local_grad_of_x = grad_of_x[rank*per_batch_size:(rank+1)*per_batch_size]
        ctx.save_for_backward(local_grad_of_x, grad_of_weights)
    return local_loss, top1

def backward(ctx, grad_out1, grad_out2):
    local_grad_of_x, weight_grad = ctx.saved_variables
    return local_grad_of_x*grad_out1, weight_grad*grad_out1, None, None
```

自动并行接口

```
class DenseNet(nn.Cell):
    def __init__(self):
        super(DenseMutMulNet, self).__init__()
        self.embedding_weight = Parameter(Tensor(12288, 128))
        self.embedding = P.MatMul()
        self.fc1 = nn.Dense(128, 768, activation='relu')
        self.fc2 = nn.Dense(128, 768, activation='relu')
        self.fc3 = nn.Dense(128, 768, activation='relu')
        self.transpose = P.Transpose()
        self.matmul1 = P.MatMul()
        self.matmul2 = P.MatMul()

    def construct(self, x):
        x = self.embedding(x, self.embedding_weight)
        q = self.fc1(x)
        k = self.fc2(x)
        v = self.fc3(x)
        k = self.transpose(k, (1, 0))
        c = self.matmul1(q, k)
        s = self.matmul2(c, v)
        return s

def train_step():
    context.set_auto_parallel_context(parallel_mode=ParallelMode.AUTO_PARALLEL)
    input = Tensor(np.ones([32, 128]).astype(np.float32))
    label = Tensor(np.zeros([32, 768]).astype(np.float32))
    net = DenseNet()
    model = Model(net, opt, loss)
    train(net, input, label)
```

单机模型代码

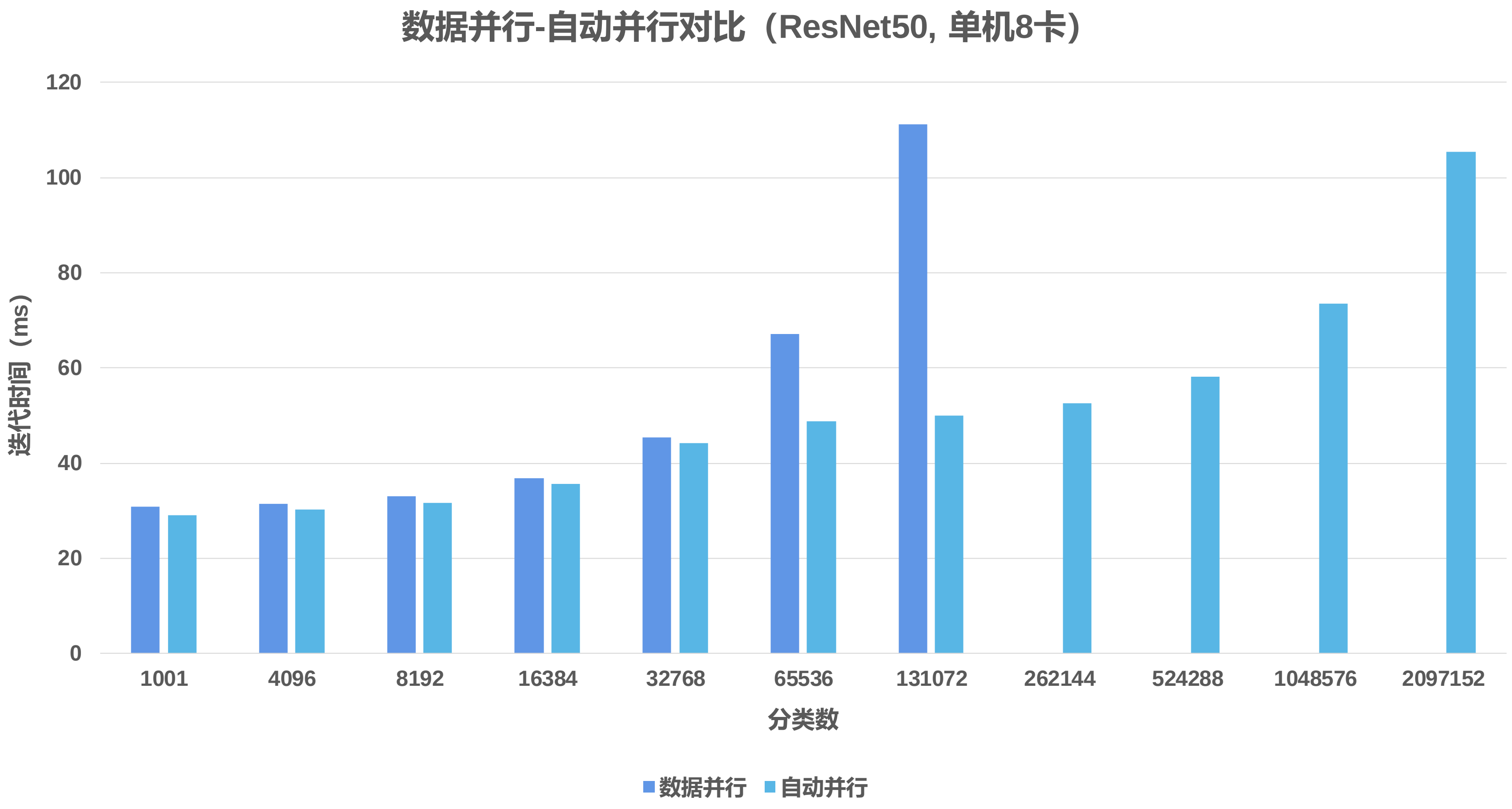
Auto Parallel

Live Demo

ResNet50网络数据并行、自动并行分布式训练编程实践

https://www.mindspore.cn/tutorial/zh-CN/master/advanced_use/distributed_training.html

性能对比



数据并行-自动并行对比（ResNet50, 单机8卡）
120
100
80
60
40
20
0
迭代时间（ms）
1001
4096
8192
16384
32768
65536
131072
262144
524288
1048576
2097152
分类数
数据并行
自动并行

课程总结

1、并行模式

- 数据并行：分发数据集，提升训练效率
- 模型并行：拆分模型参数，突破单卡内存瓶颈
- 混合并行：融合数据并行、模型并行的并行模式
- 集合通信：实现多卡间数据同步操作

2、自动并行

- 算子自动切分，推导张量排布模型
- 构建代价模型，自动搜索切分策略
- 将自动微分扩展到分布式领域，自动插入通信算子

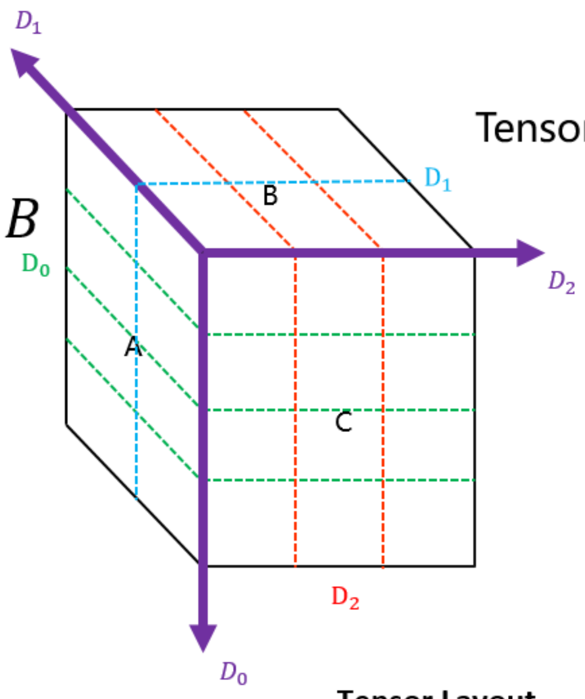
3、分布式训练

- 支持数据并行、模型并行、自动并行多层次接口
- 自动并行更加易用，帮助网络提升性能

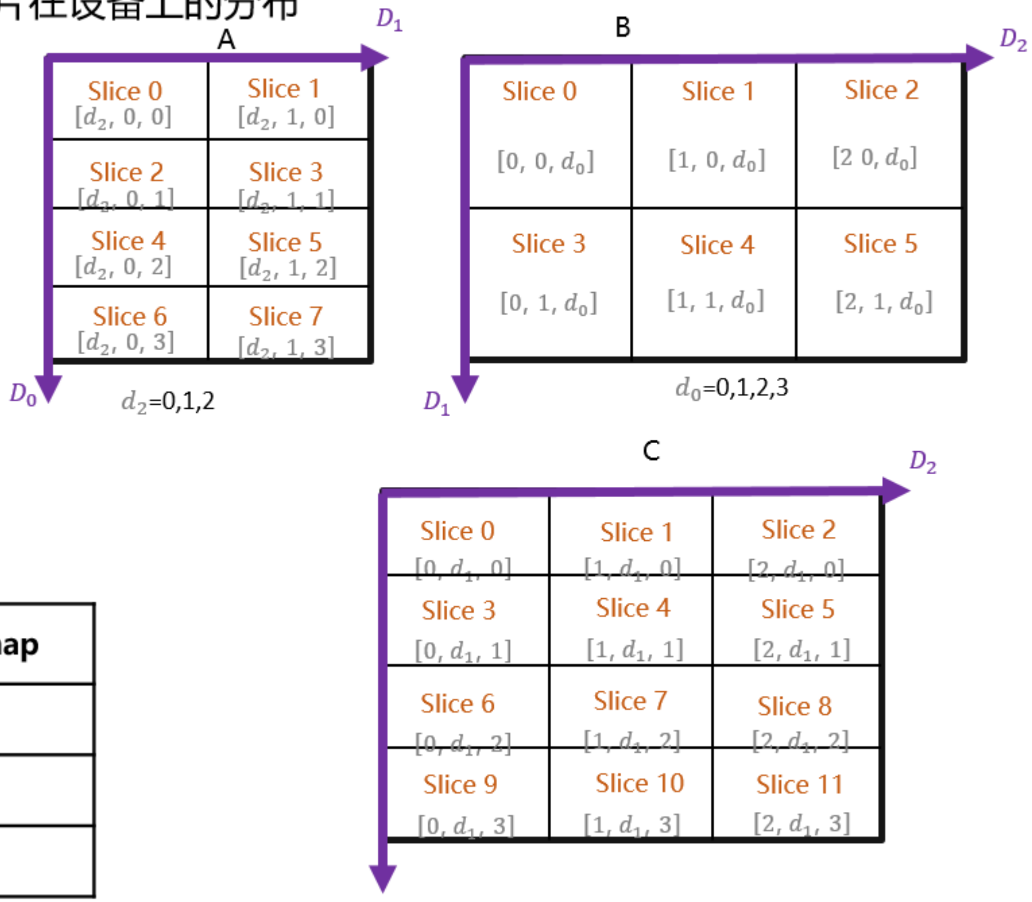
THANK YOU

张量排布模型

$$C = A \times B$$



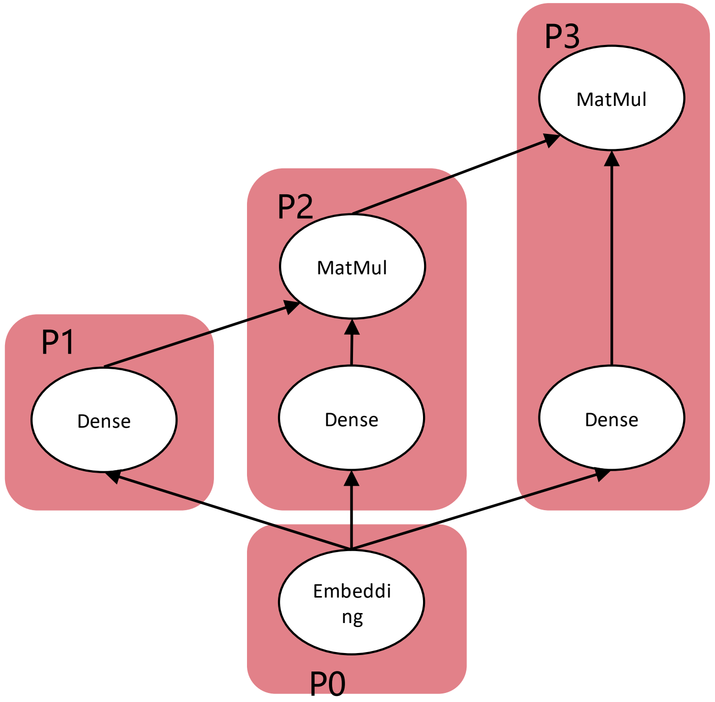
Tensor切片在设备上的分布



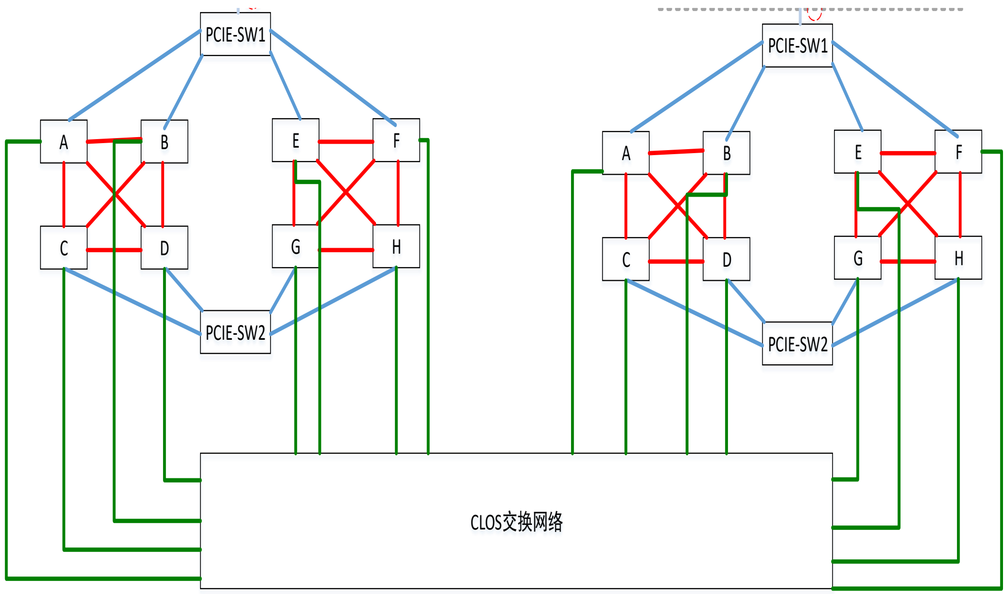
Tensor Layout

Tensor	device_matrix	tensor_map
A	$[D_2, D_1, D_0]$ $D_2=3$ $D_1=2$ $D_0=4$	[0, 1]
B		[1, 2]
C		[0, 2]

手动切分的难度



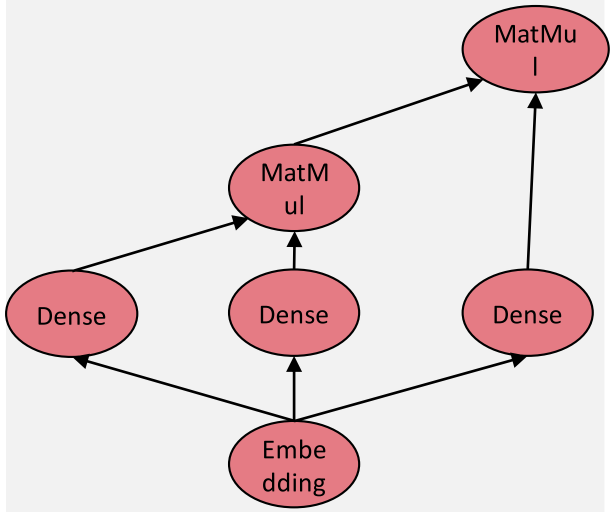
- 内存上限
- 计算均衡
- 网络拓扑



MindSpore自动并行架构

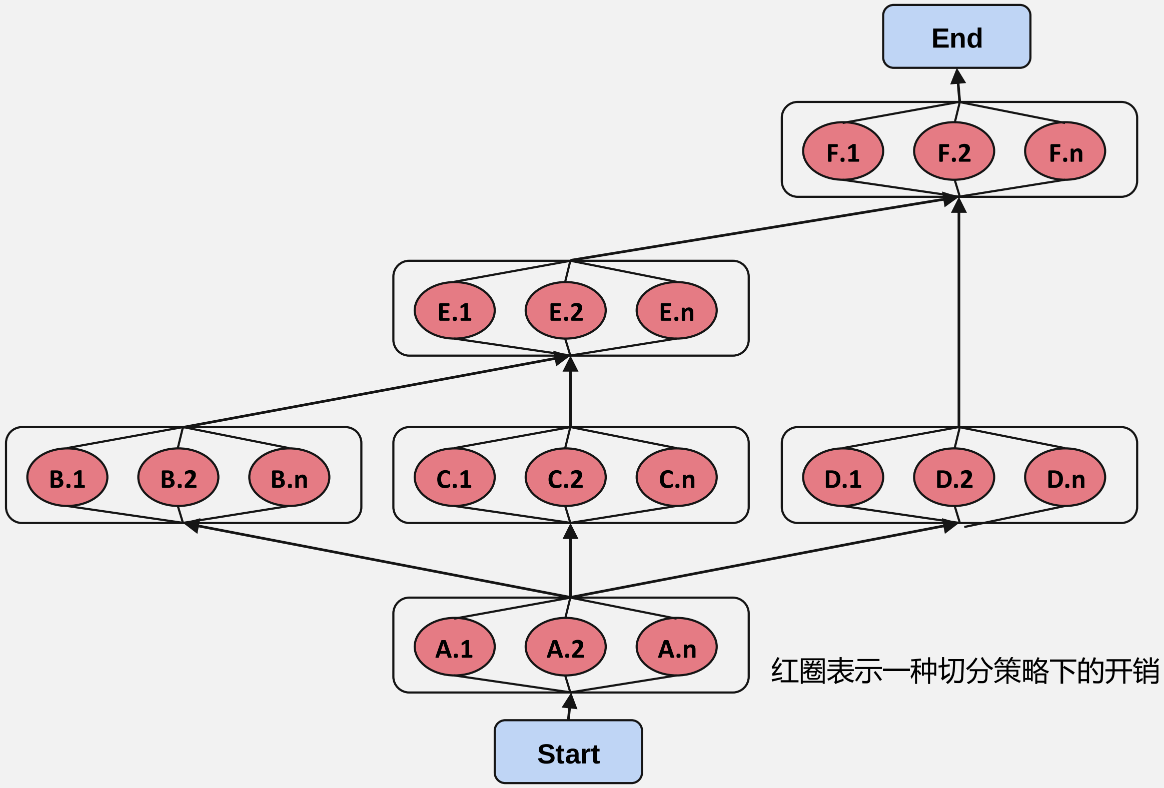
MindSpore

NP-hard：在内存上限约束下，搜索出性能较优（计算时间+通信时间）的切分策略



模型代码构造出ANF图

Cost Model



ANF图构造出Cost Model图，在图上搜索出较优策略

联系我们

如果您有任何问题，欢迎关注我们的gitee和github，提issue，我们会即使为您解答。您也可以加入我们的官方QQ群，直接与技术专家互动~

官网：https://www.mindspore.cn/
Gitee：https://gitee.com/mindspore
GitHub：https://github.com/mindspore-ai

MindSpore官方交流群
扫一扫二维码，加入群聊。