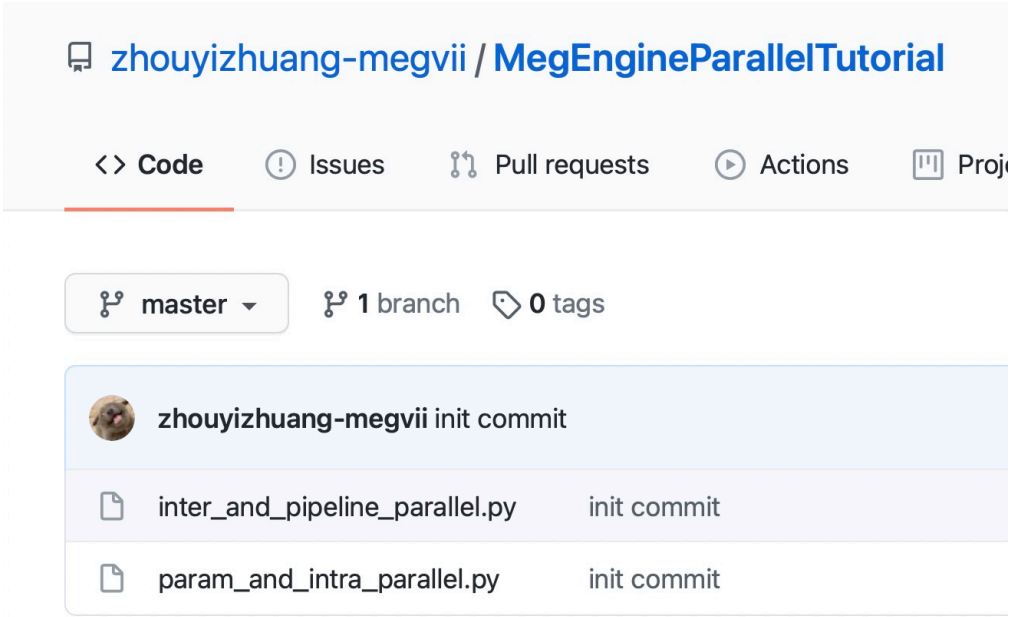
利用MegEngine的分布式通信算子实现复杂的并行训练

MEGVII 旷视

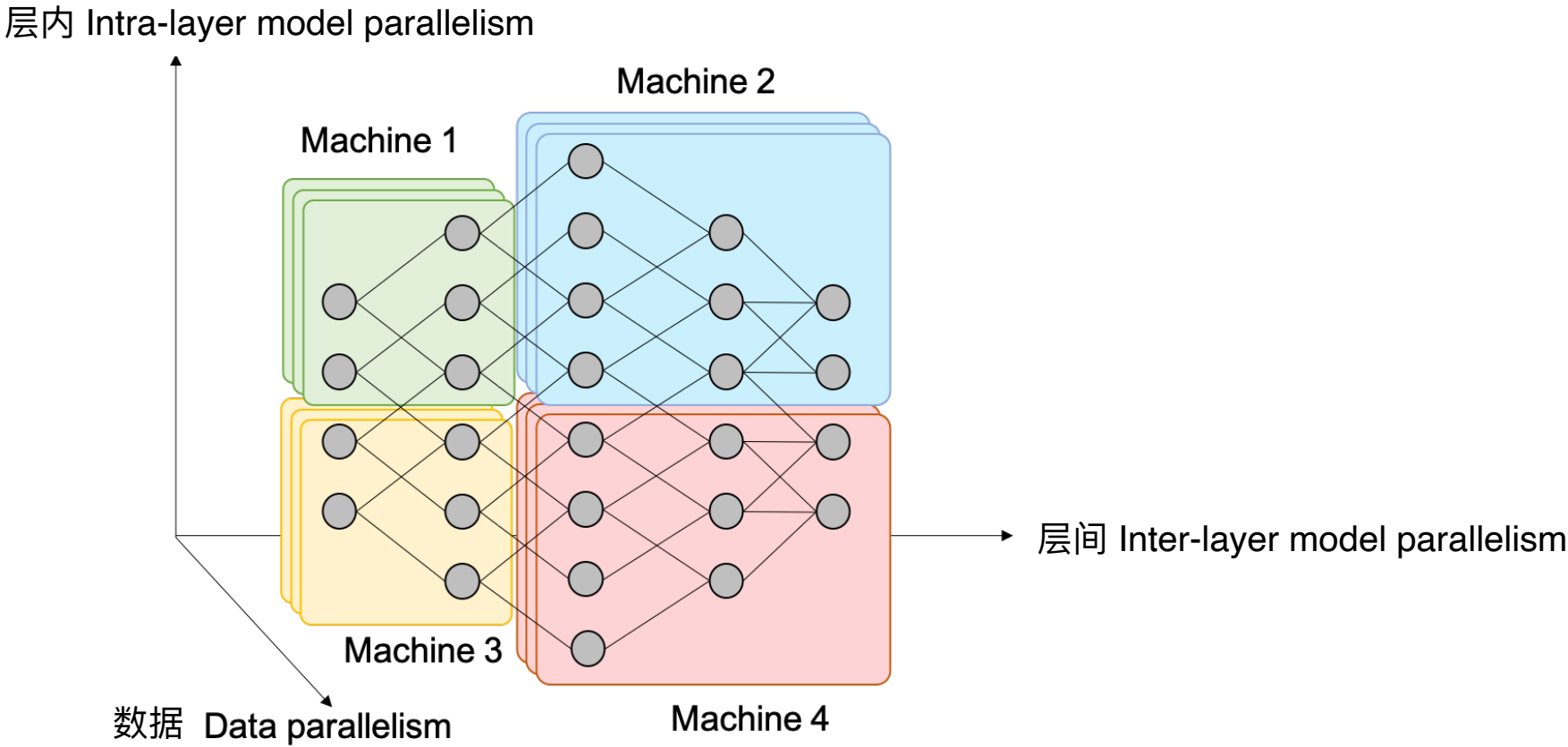
旷视研究院基础模型组研究员 周亦庄

2021年03月25日



Code Available:

https://github.com/zhouyizhuang-megvii/MegEngineParallelTutorial


层内 Intra-layer model parallelism
Machine 2
Machine 1
Machine 3
Machine 4
层间 Inter-layer model parallelism
数据 Data parallelism

目录

Contents

0. MegEngine的通信算子

1. 简单参数并行

2. 层内模型并行(Intra-layer Model Parallelism)

    2.1. 全连接(Fully Connected)模型并行

    2.2. 组卷积(Group Convolution)模型并行

3. 层间模型并行(Inter-layer Model Parallelism)

    3.1. 简单模型并行

    3.2. 流水线并行(Pipeline Parallelism)

        3.2.1. 流水线推理

        3.2.2. 手动checkpoint与GPipe

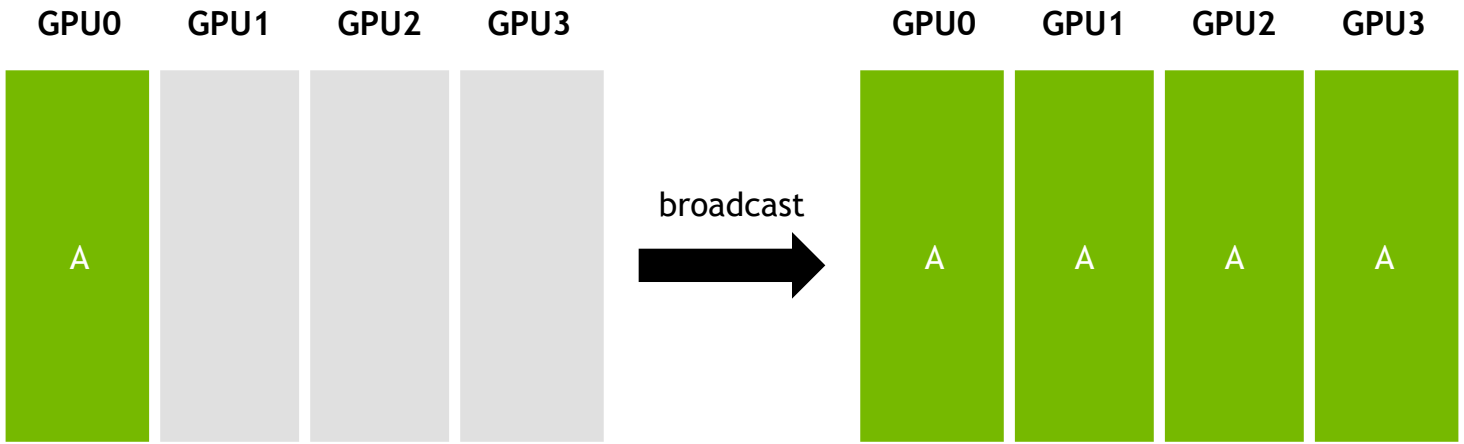
0. MegEngine的通信算子

MEGVII 旷视

F.distributed.__	算子	对应的微分算子	行为	常用用途
Collective communication 集合通信（一对多）	broadcast	reduce_sum	同步	数据并行
	reduce_sum	broadcast	求和	
	scatter	gather	拆分	
	gather	scatter	集中	
Collective Communication 集合通信（多对多）	all_reduce_sum	all_reduce_sum	同步求和	数据并行
	all_gather	reduce_scatter_sum	同步集中	层内模型并行
	reduce_scatter_sum	all_gather	求和后拆分	层内模型并行
	all_to_all	all_to_all	转置	层内模型并行
Point-to-point Communication 点对点通信	remote_send	remote_recv	发送	层间模型并行
	remote_recv	remote_send	接收	层间模型并行

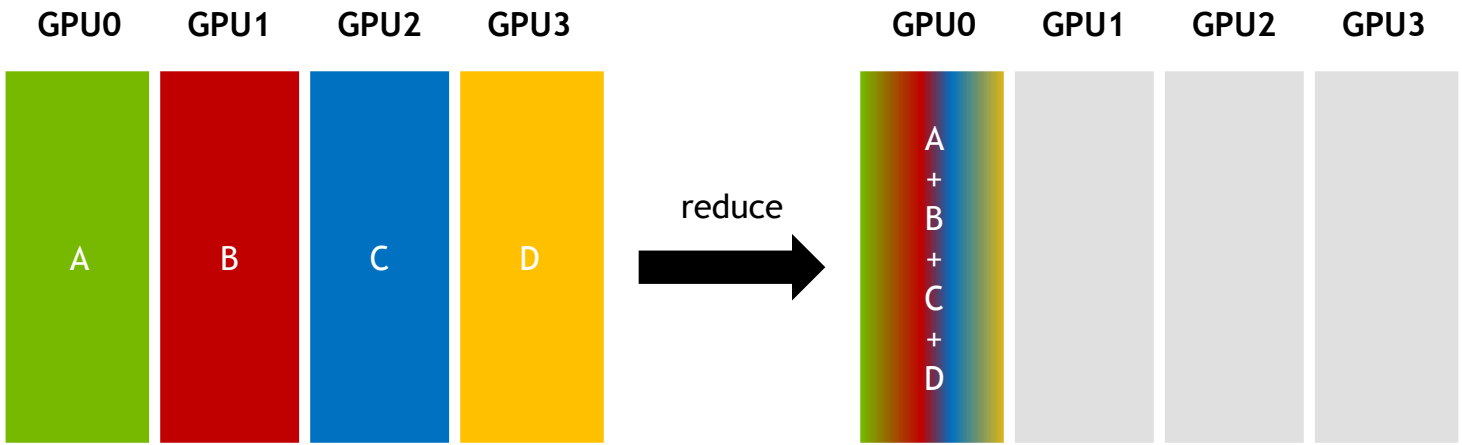
BROADCAST

One sender, multiple receivers



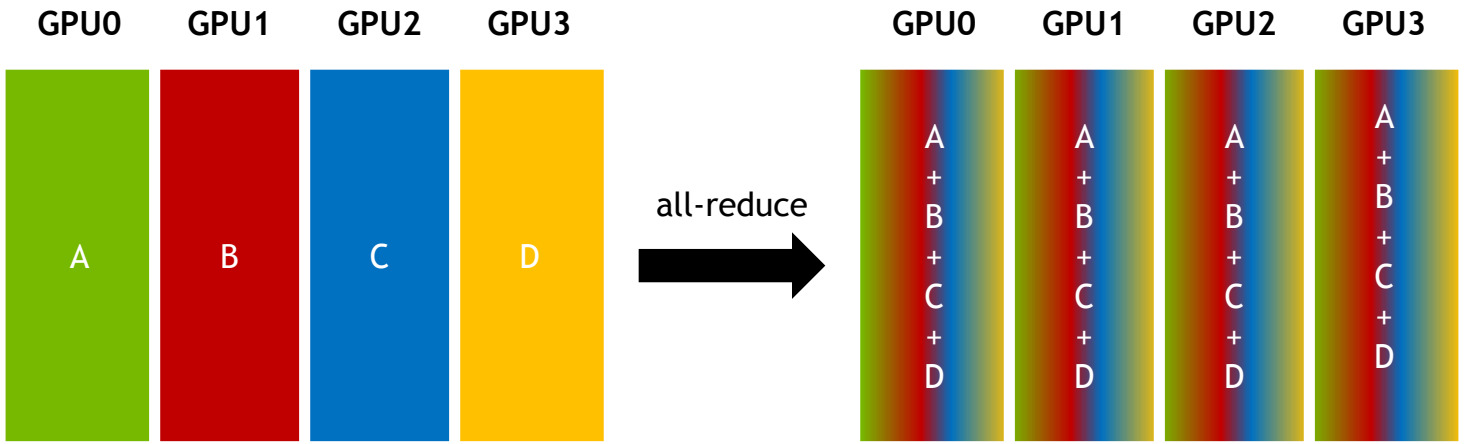
REDUCE

Combine data from all senders; deliver the result to one receiver



ALL-REDUCE

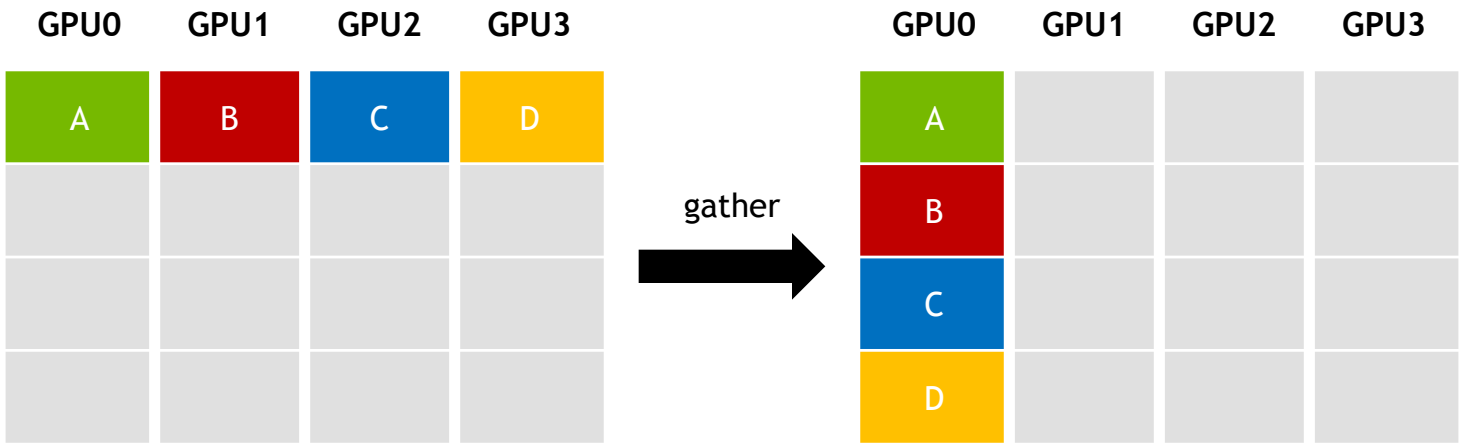
Combine data from all senders; deliver the result to all participants



MEGVII 旷视

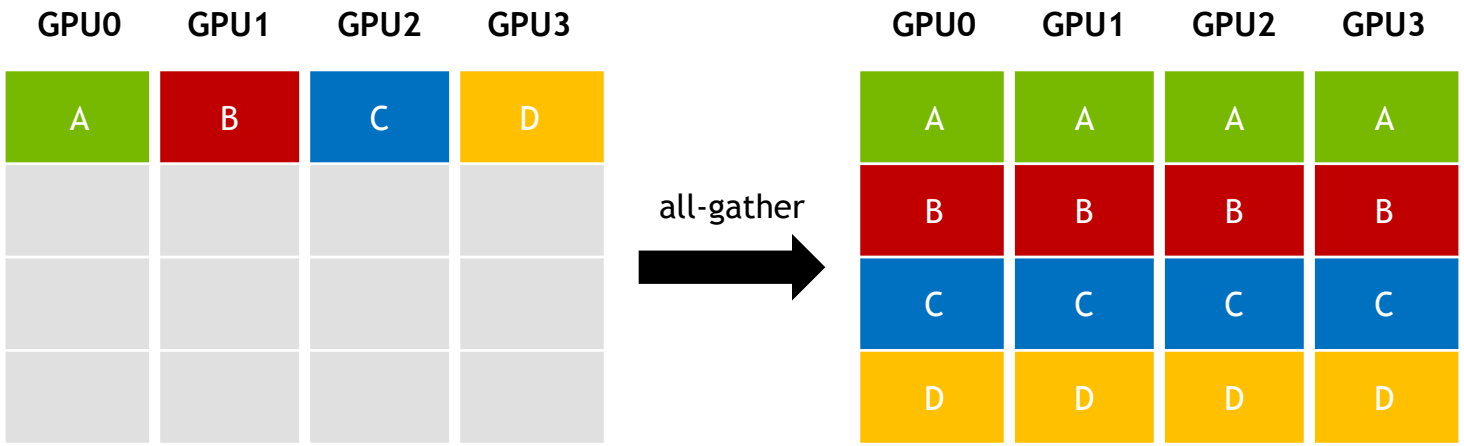
GATHER

Multiple senders, one receiver



ALL-GATHER

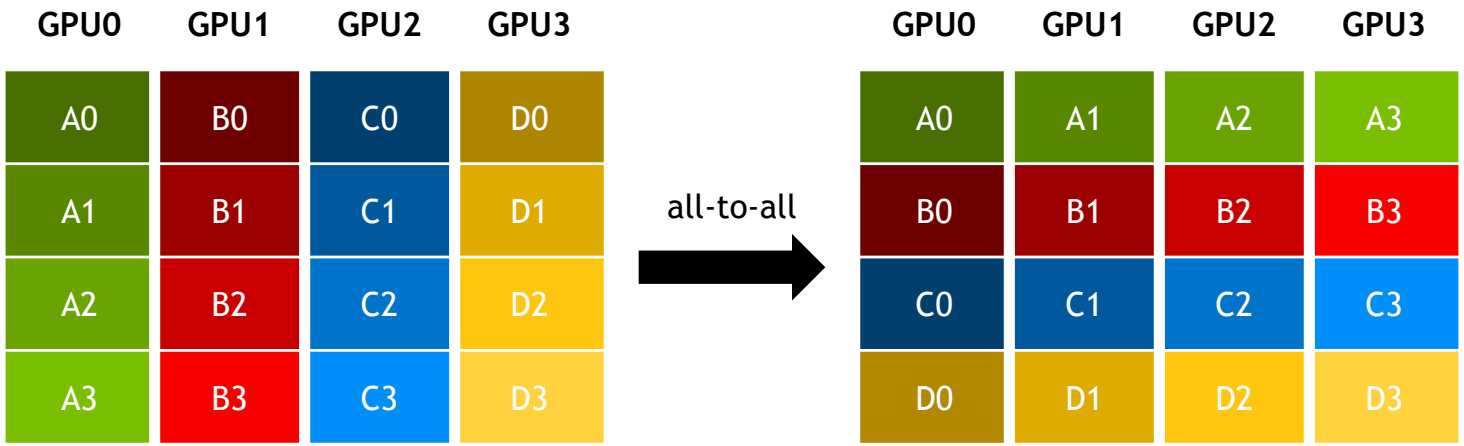
Gather messages from all; deliver gathered data to all participants



MEGVII 旷视

ALL-TO-ALL

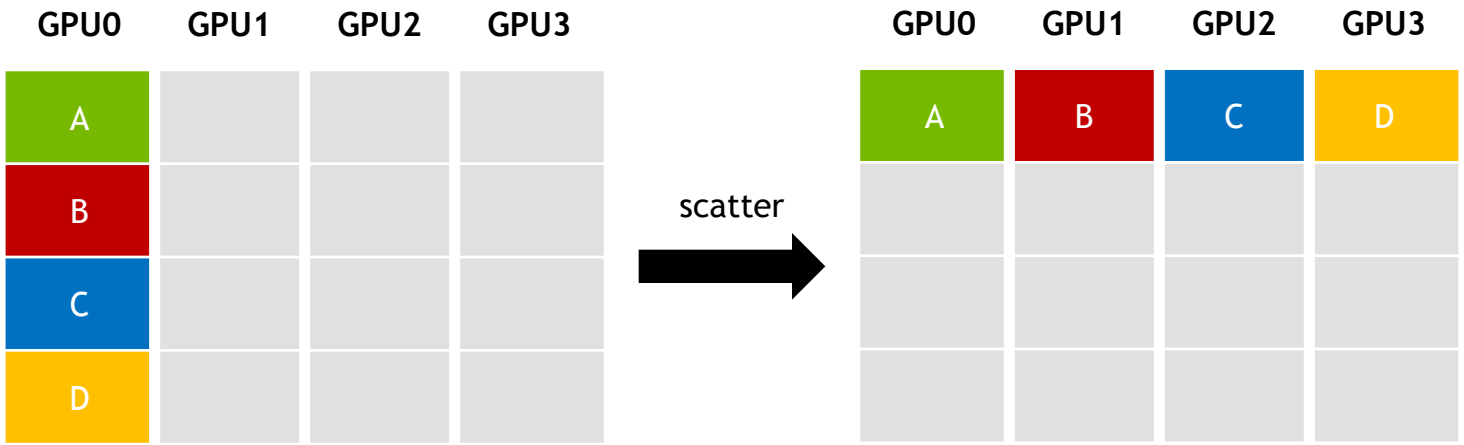
Scatter/Gather distinct messages from each participant to every other



MEGVII 旷视

SCATTER

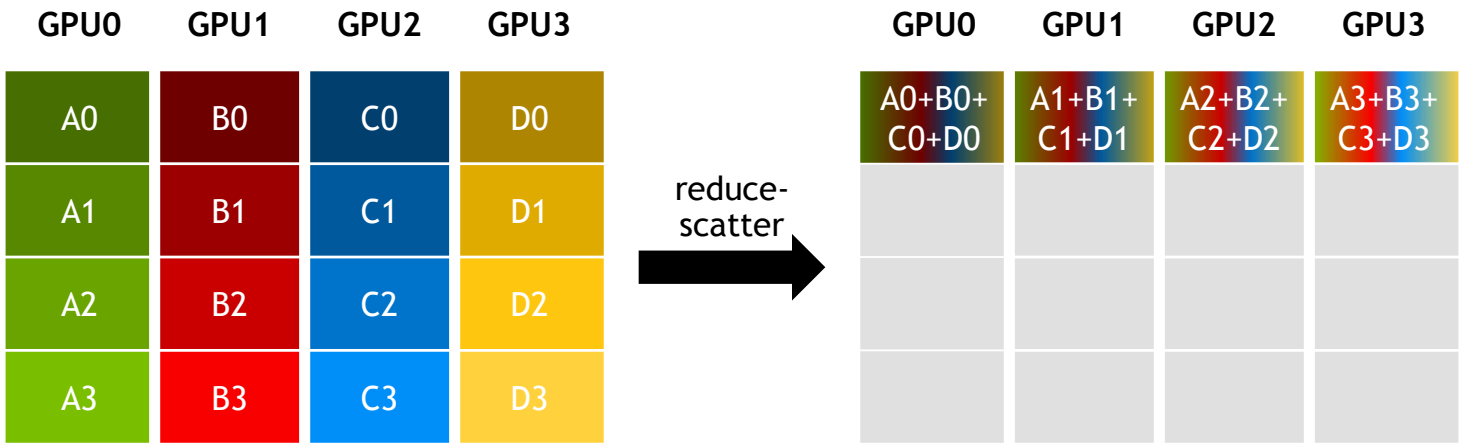
One sender; data is distributed among multiple receivers



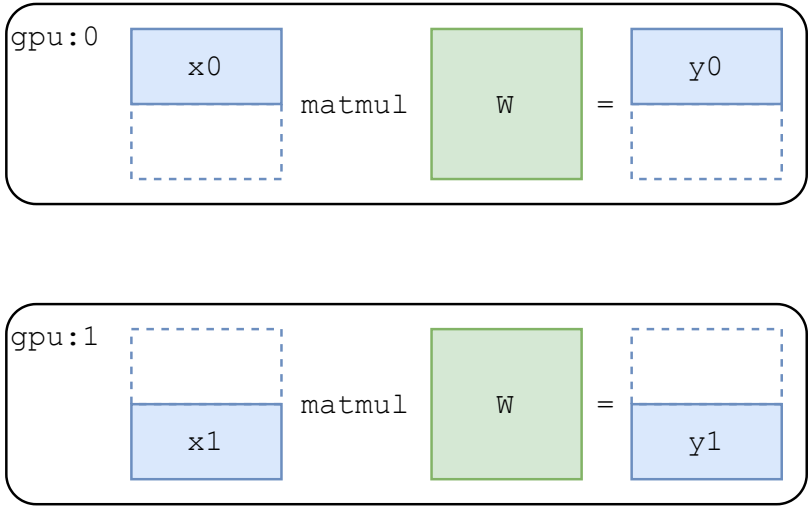
MEGVII 旷视

REDUCE-SCATTER

Combine data from all senders; distribute result across participants

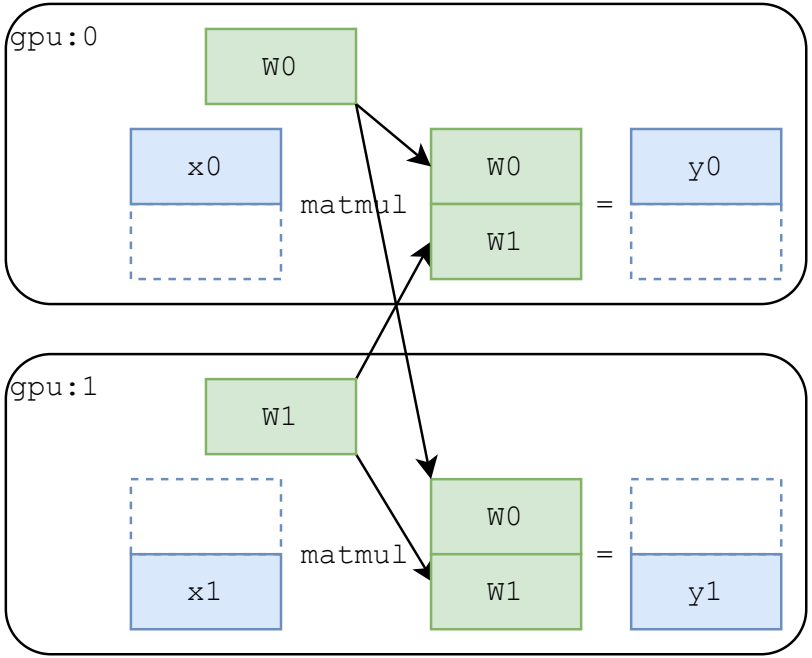


1. 数据并行



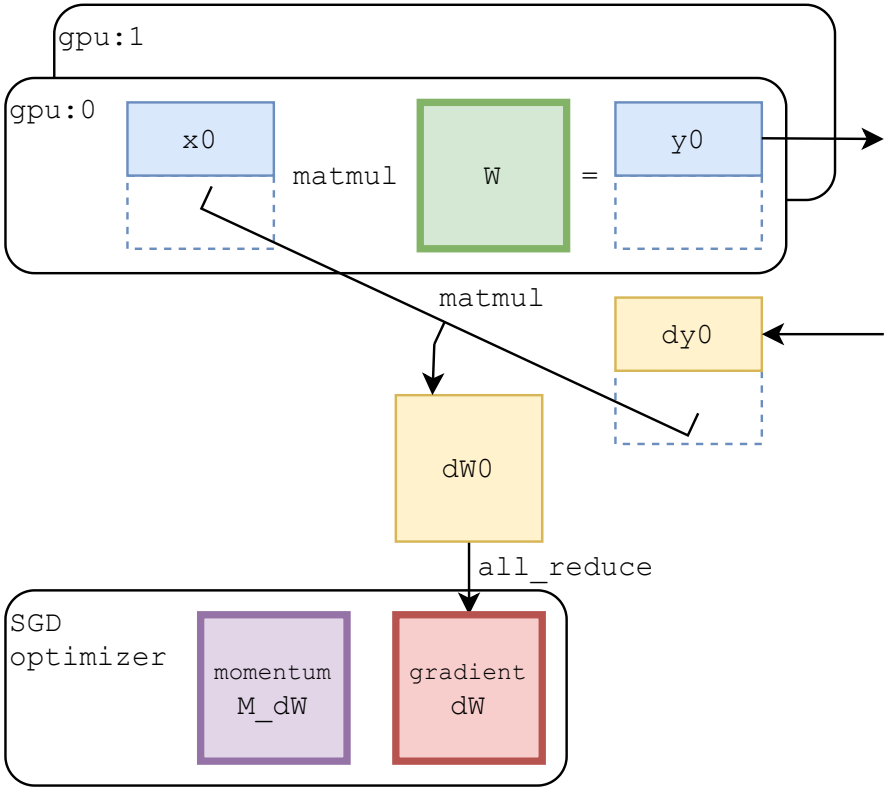

```
def forward(self, x):
    return F.nn.linear(x, self.weight)
```

2. 简单参数并行

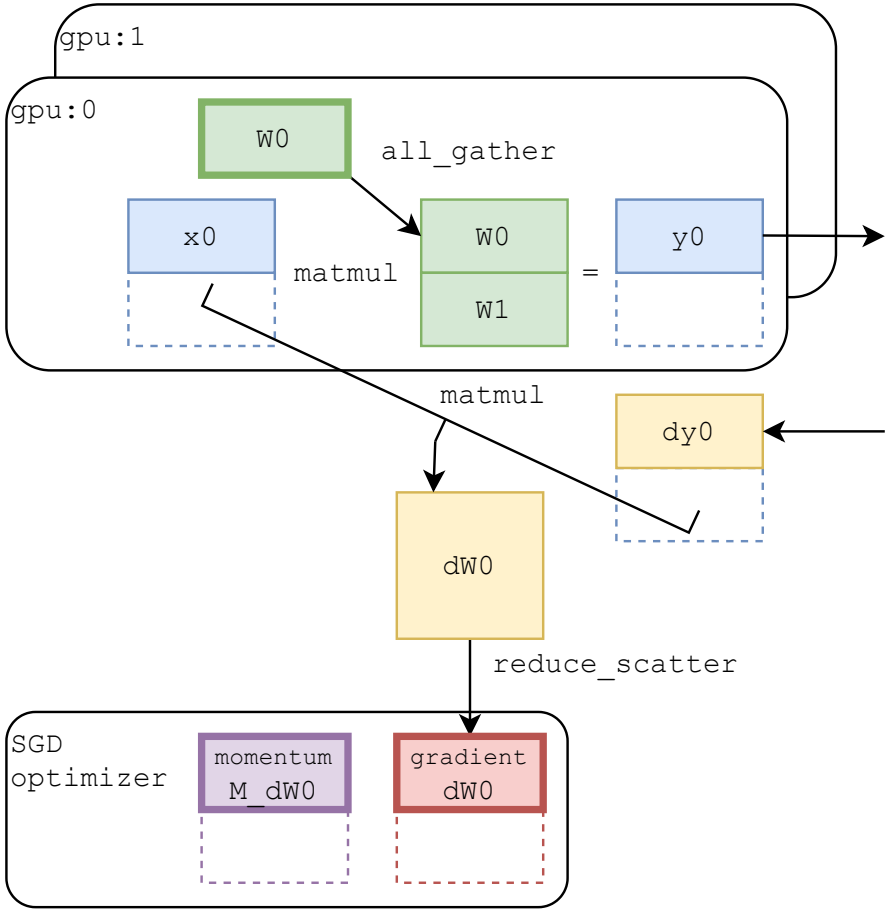



```
def forward(self, x):
    weight = F.distributed.all_gather(self.weight)
    return F.nn.linear(x, weight)
```

1．数据并行



2．简单参数并行



1. 简单参数并行

MEGVII 旷视

1. 数据并行

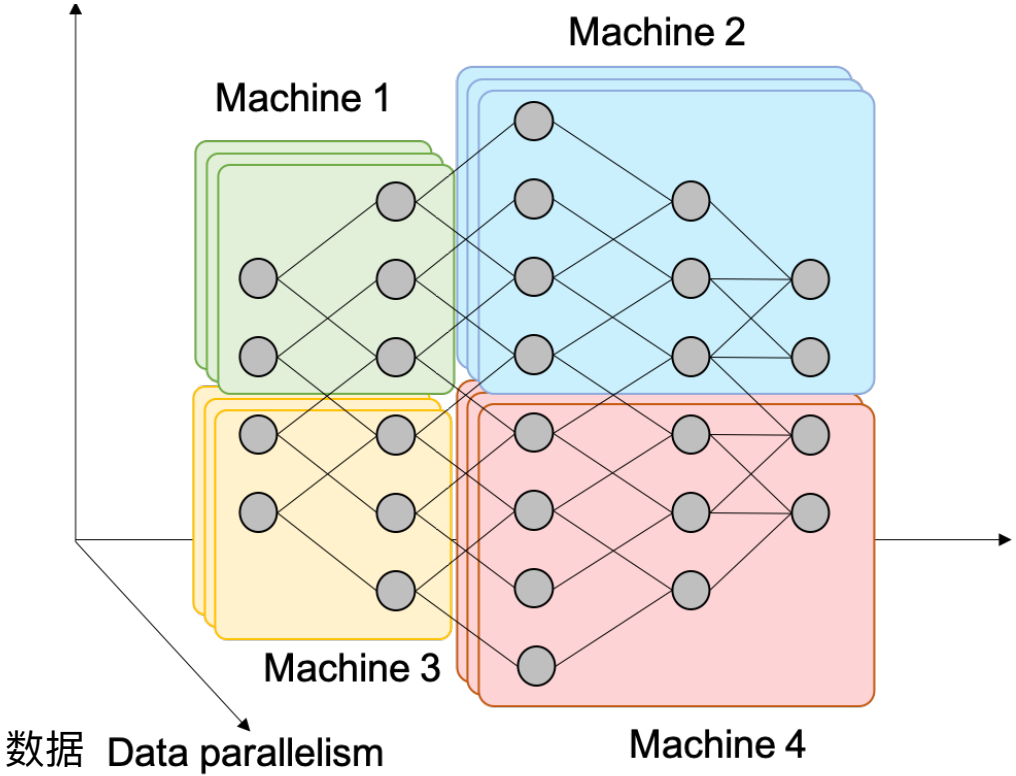
```
model = M.Linear(inp, oup, bias=False)

def forward(self, x):
    return F.nn.linear(x, self.weight)


model.forward = forward

# equal to `dist.bcast_list_(model.parameters())`
for p in model.parameters():
    p._reset(F.distributed.broadcast(p))

gm = ad.GradManager().attach(
        model.parameters(),
        callbacks=dist.make_allreduce_cb("SUM")
)
```

2. 简单参数并行

```
model = M.Linear(inp, oup, bias=False)

def forward(self, x):
    weight = F.distributed.all_gather(self.weight)
    return F.nn.linear(x, weight)

model.forward = forward


for p in model.parameters():
    p._reset(F.distributed.scatter(p))

gm = ad.GradManager().attach(
        model.parameters(),
)
```

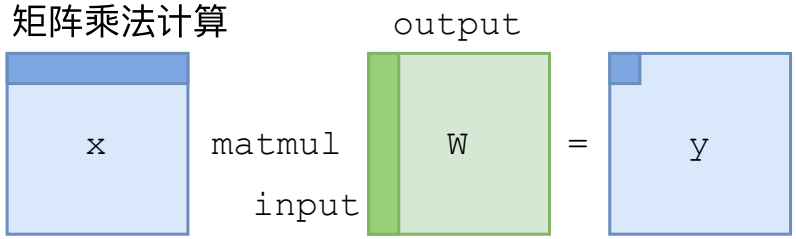
2. 层内模型并行(Intra-layer Model Parallelism)

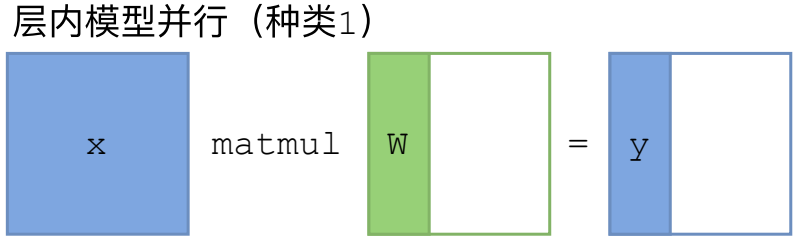
层内 Intra-layer model parallelism

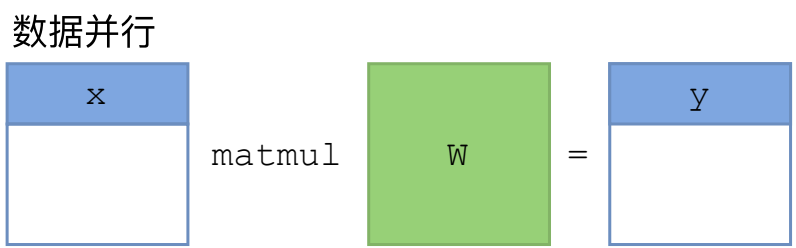


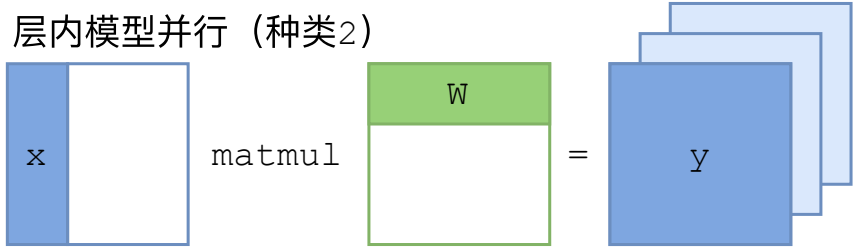
层间 Inter-layer model parallelism

2. 层内模型并行(Intra-layer Model Parallelism)





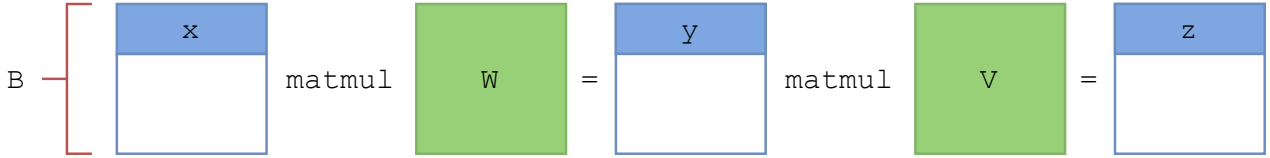




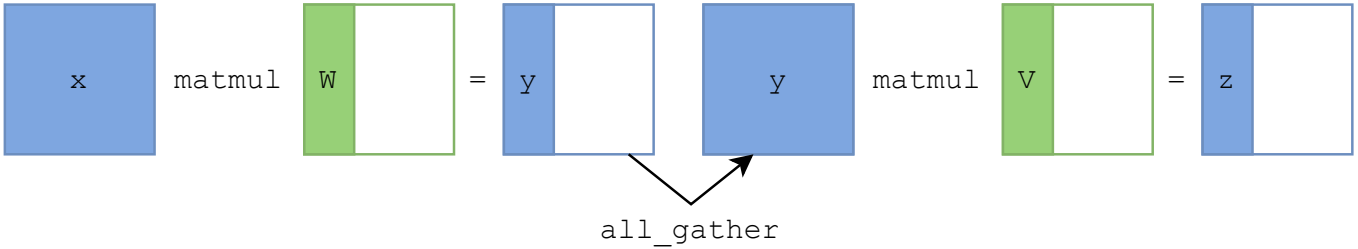
2. 层内模型并行(Intra-layer Model Parallelism)

MEGVII 旷视

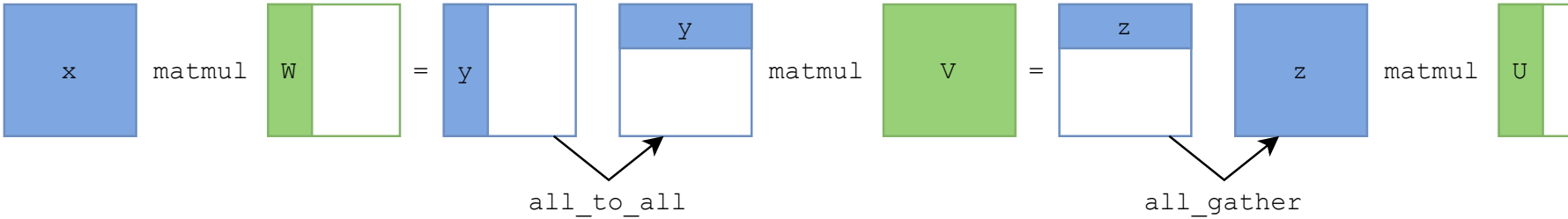
并行方式1（即数据并行）



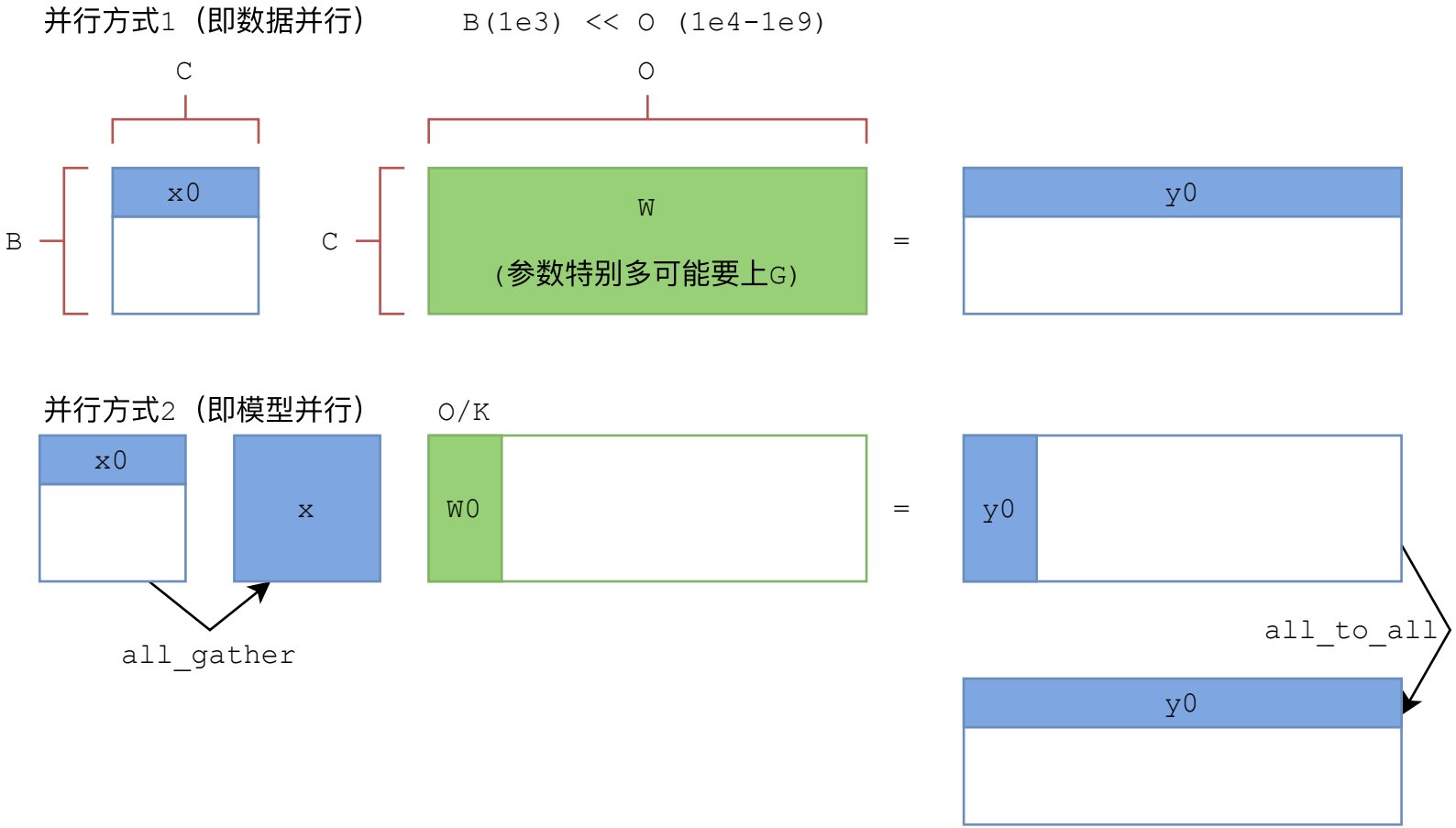
并行方式2（即模型并行）



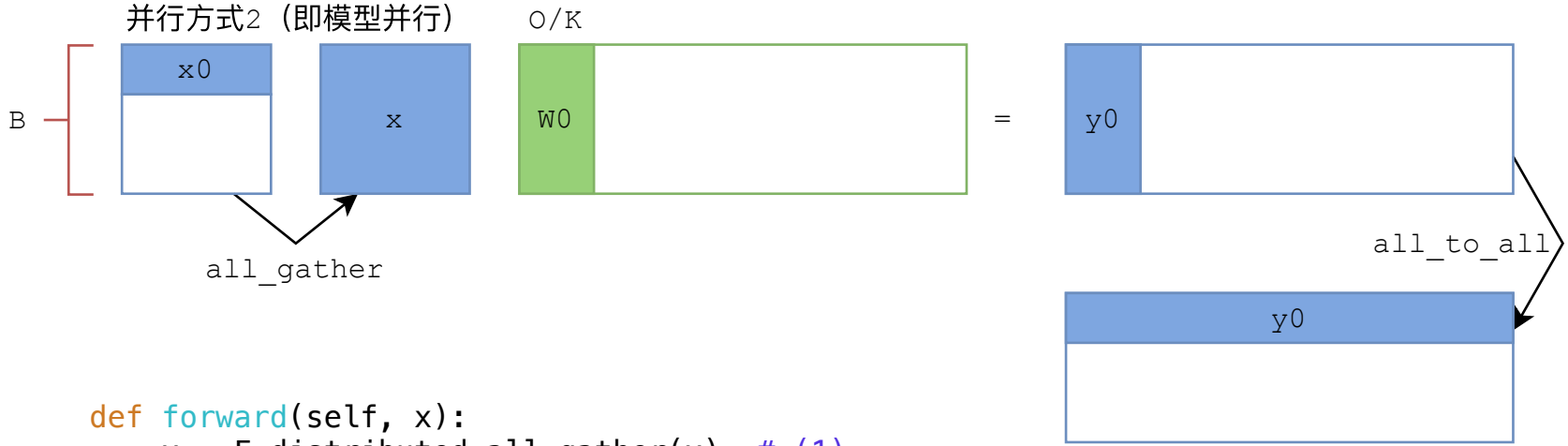
并行方式3（即混合并行）



2.1. 场景一：全连接(Fully Connected)模型并行

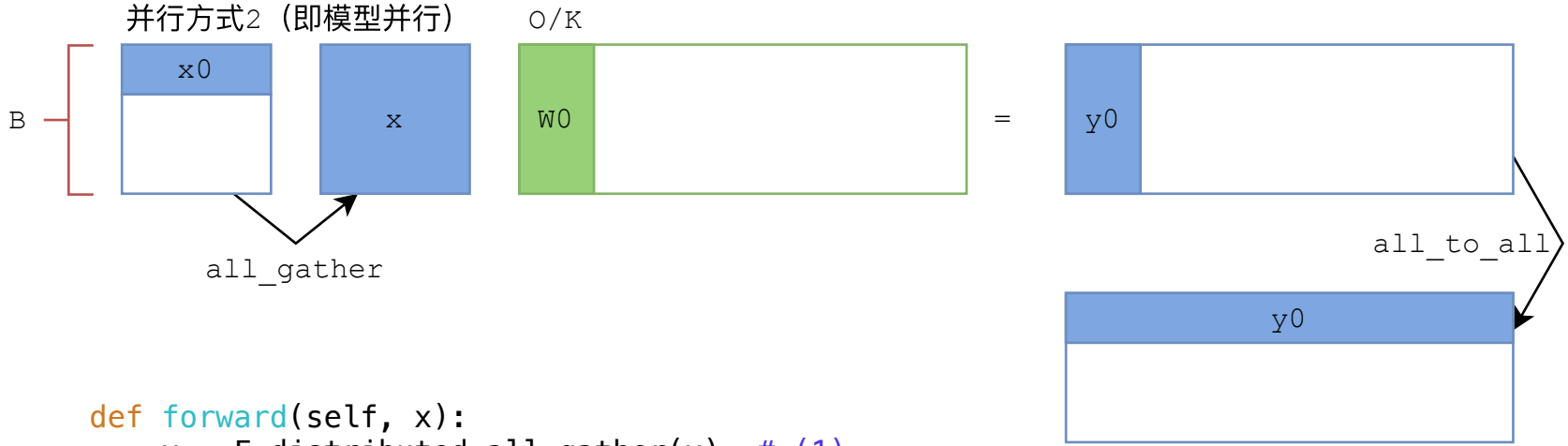


2.1. 场景一：全连接(Fully Connected)模型并行




```
def forward(self, x):
    x = F.distributed.all_gather(x)  # (1)
    y = F.nn.linear(x, self.weight)  # (2)

    y = F.distributed.all_to_all(y)  # (3)
    n, c, *shp = y.shape
    y = y.reshape(k, n // k, -1)\
         .transpose(1, 0, 2)\
         .reshape(n // k, c * k, *shp)

    return y
```

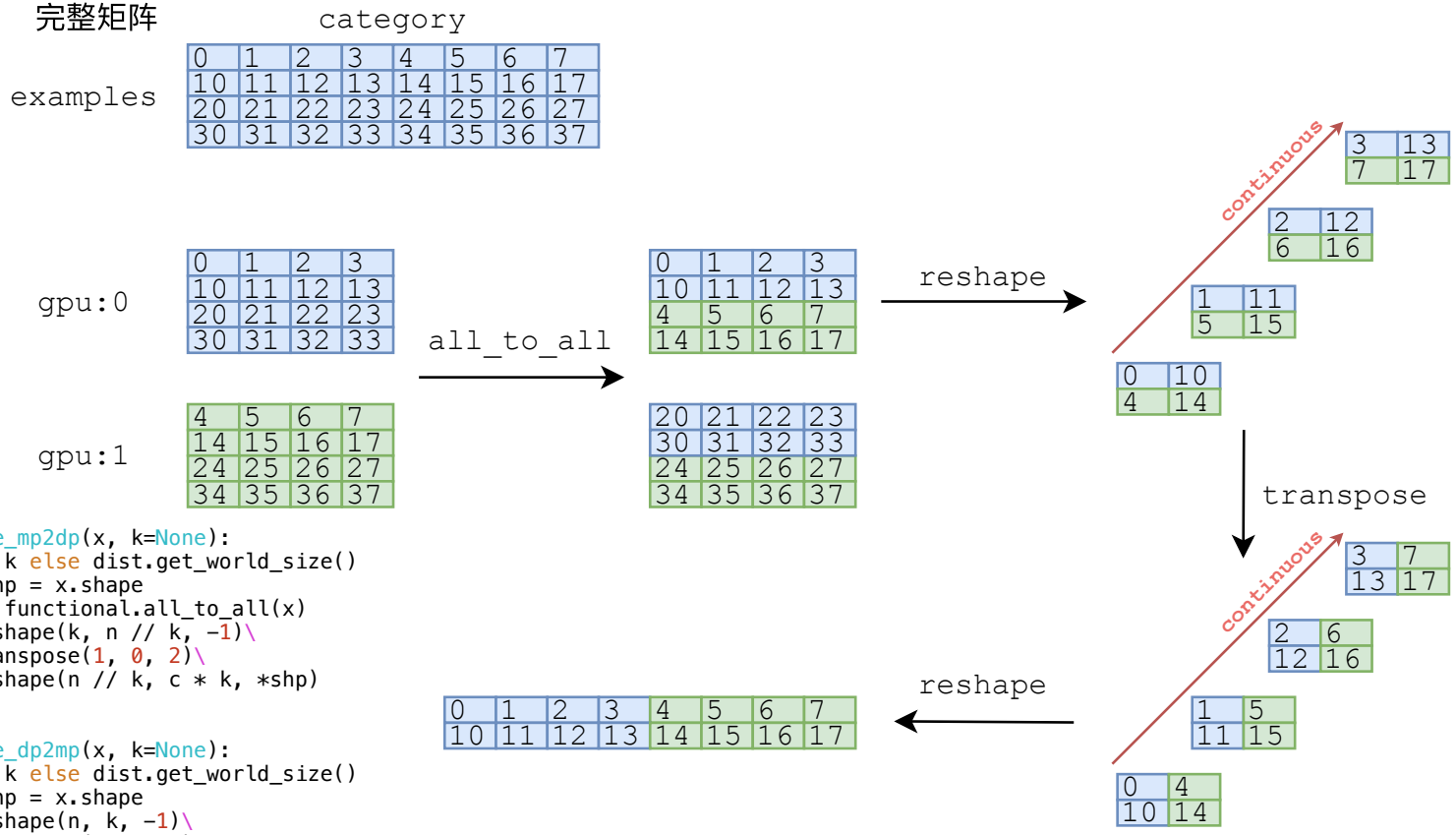
2.1. 场景一：全连接(Fully Connected)模型并行




```
def forward(self, x):
    x = F.distributed.all_gather(x)  # (1)
    y = F.nn.linear(x, self.weight)  # (2)

    y = F.distributed.all_to_all(y)  # (3)
    n, c, *shp = y.shape
    y = y.reshape(k, n // k, -1)\
         .transpose(1, 0, 2)\
         .reshape(n // k, c * k, *shp)

    return y
```

reshape...transpose...
这些是什么？？
数据重排布！

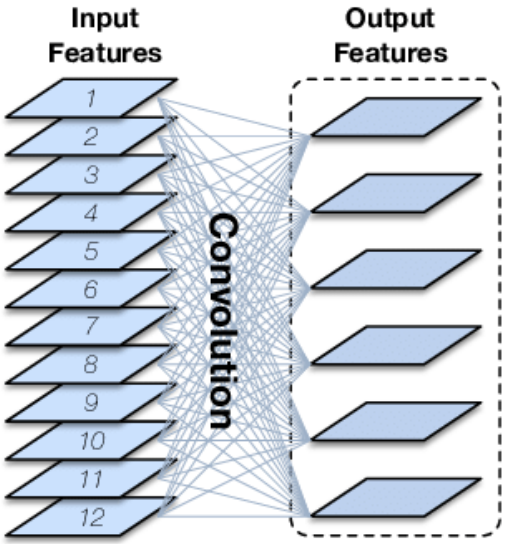
2.1. 场景一：全连接(Fully Connected)模型并行



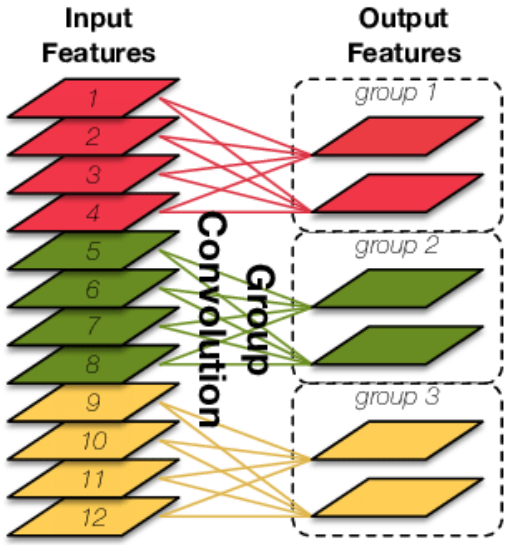

```
def transpose_mp2dp(x, k=None):
    k = k if k else dist.get_world_size()
    n, c, *shp = x.shape
    x = dist.functional.all_to_all(x)
    x = x.reshape(k, n // k, -1)\
         .transpose(1, 0, 2)\
         .reshape(n // k, c * k, *shp)
    return x

def transpose_dp2mp(x, k=None):
    k = k if k else dist.get_world_size()
    n, c, *shp = x.shape
    x = x.reshape(n, k, -1)\
         .transpose(1, 0, 2)\
         .reshape(n * k, c // k, *shp)
    x = dist.functional.all_to_all(x)
    return x
```

2.2. 场景二：组卷积(Group Convolution)模型并行

普通卷积

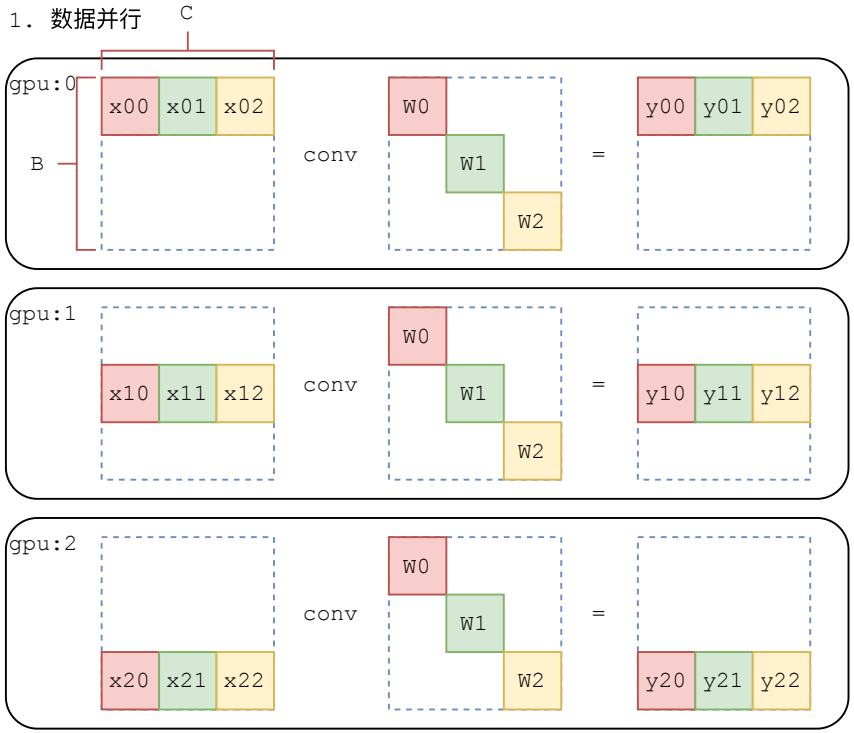


组卷积（等价于K个普通卷积）

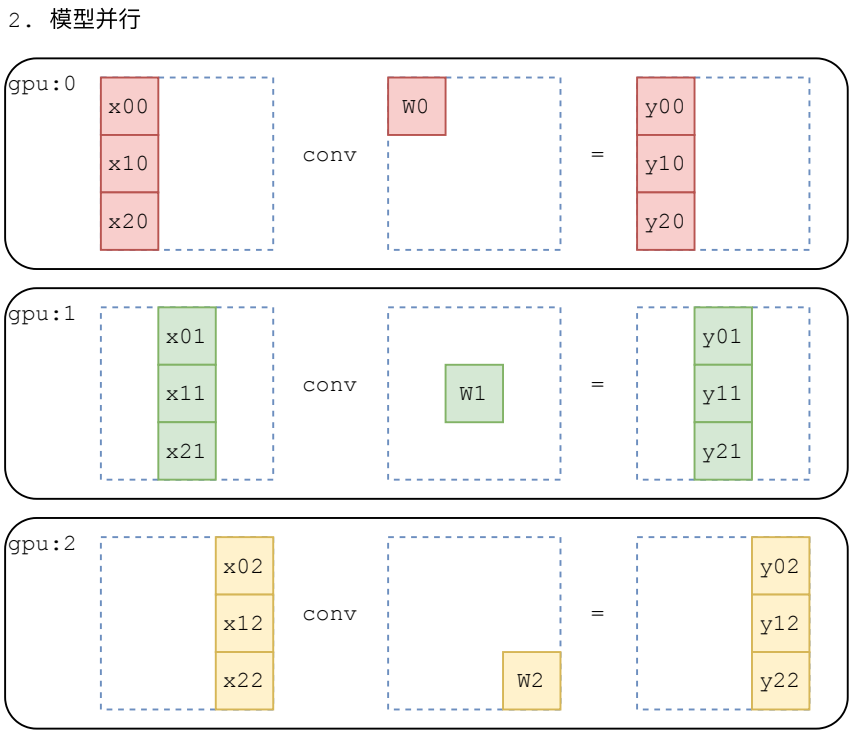


2.2. 场景二：组卷积(Group Convolution)模型并行

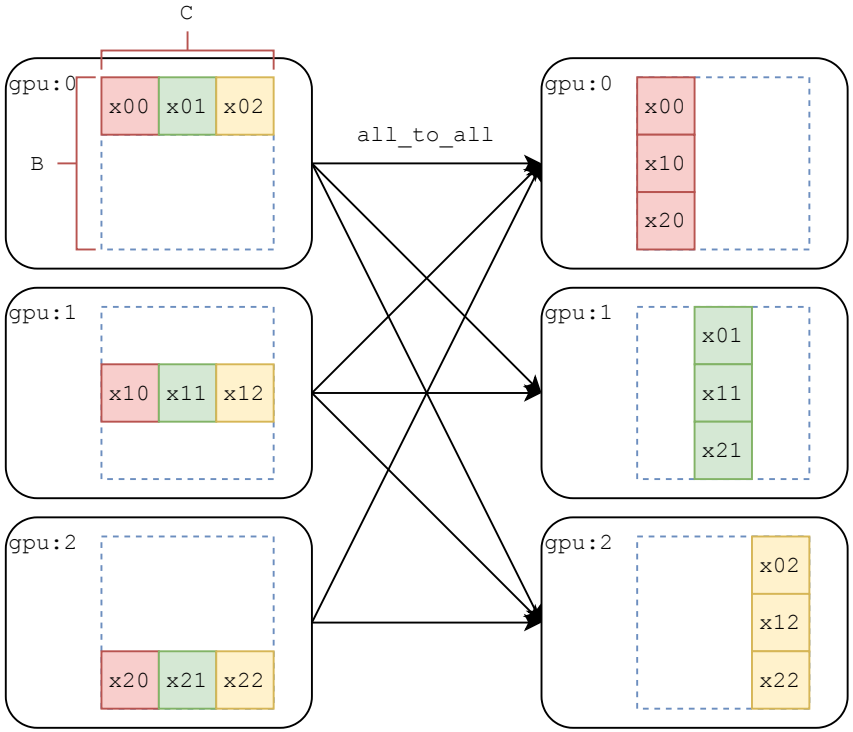
1．数据并行



2．模型并行

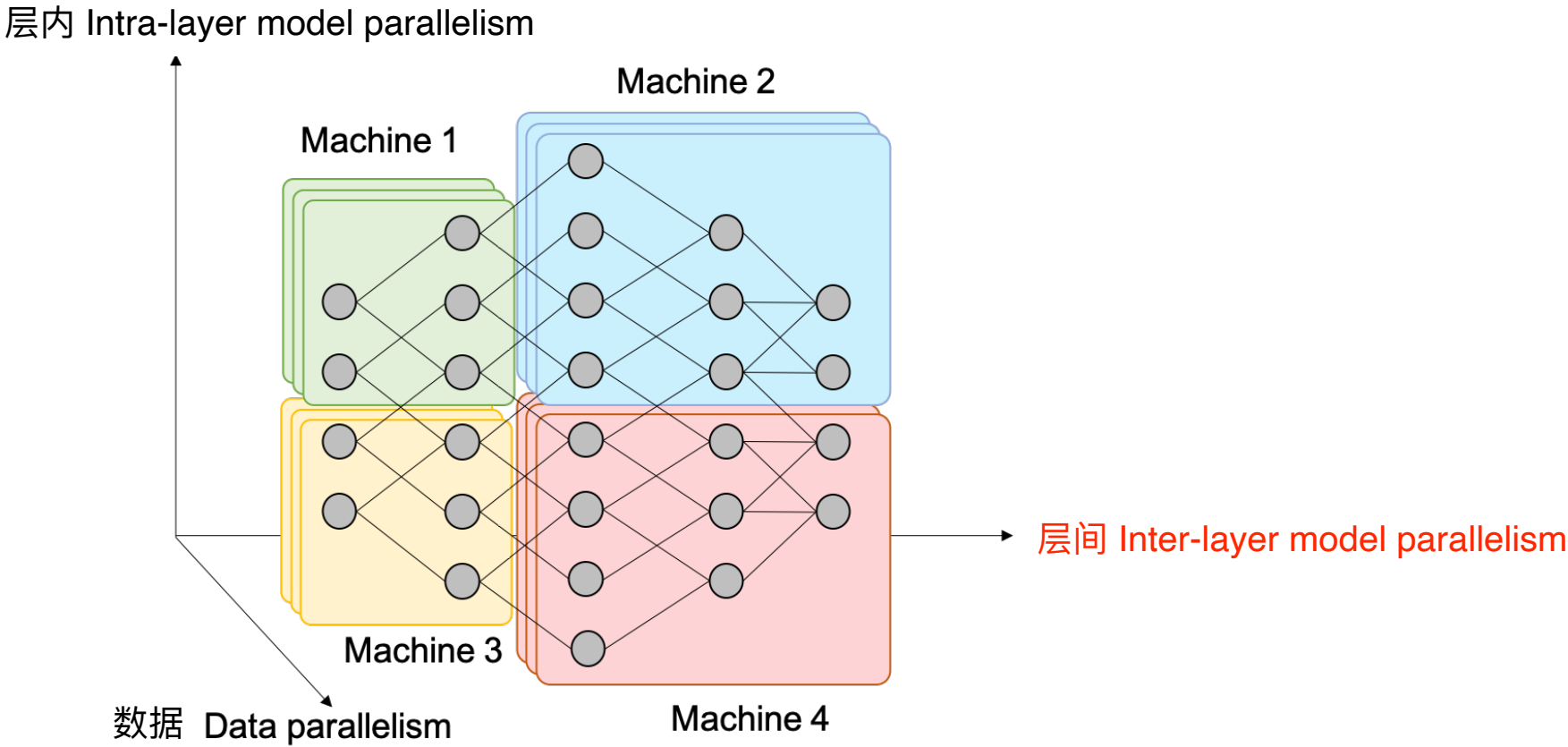


2.2. 场景二：组卷积(Group Convolution)模型并行




```python
def forward(self, x, is_head=False, is_tail=False):
    k = dist.get_world_size()
    if is_head:
        x = transpose_dp2mp(x, k=k)

    x = F.conv2d(x, self.weight, groups=1, **kwargs)

    if is_tail:
        x = transpose_mp2dp(x, k=k)

    return x
```

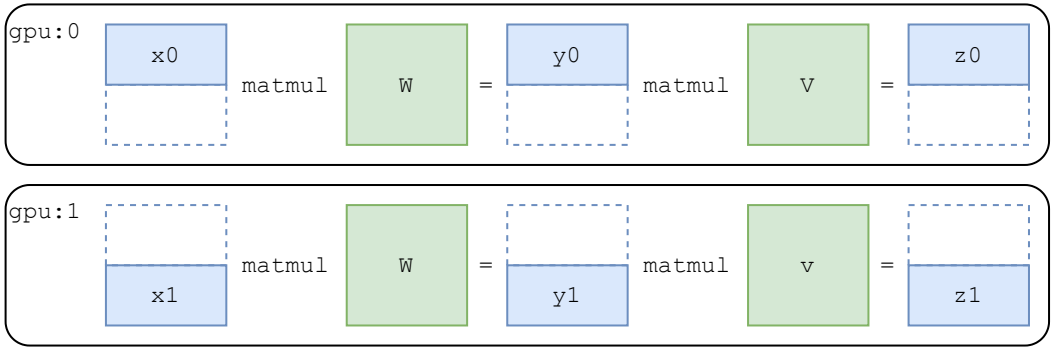
3. 层间模型并行(Inter-layer Model Parallelism)



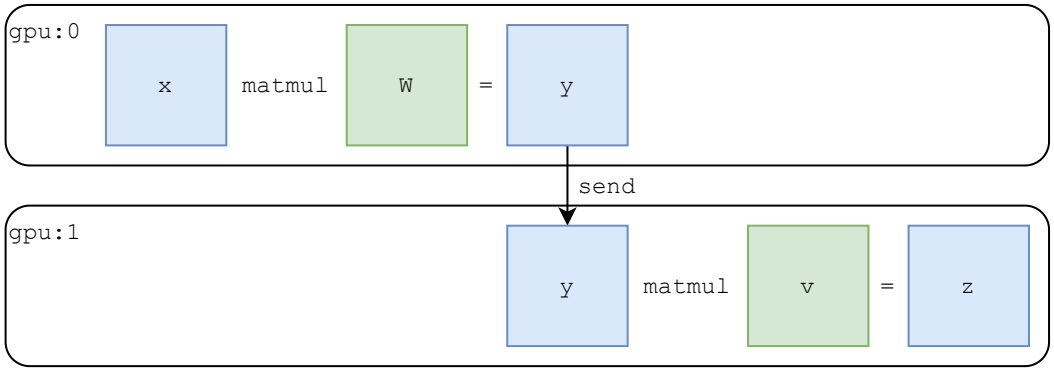
3. 层间模型并行(Inter-layer Model Parallelism)

MEGVII 旷视

1. 数据并行



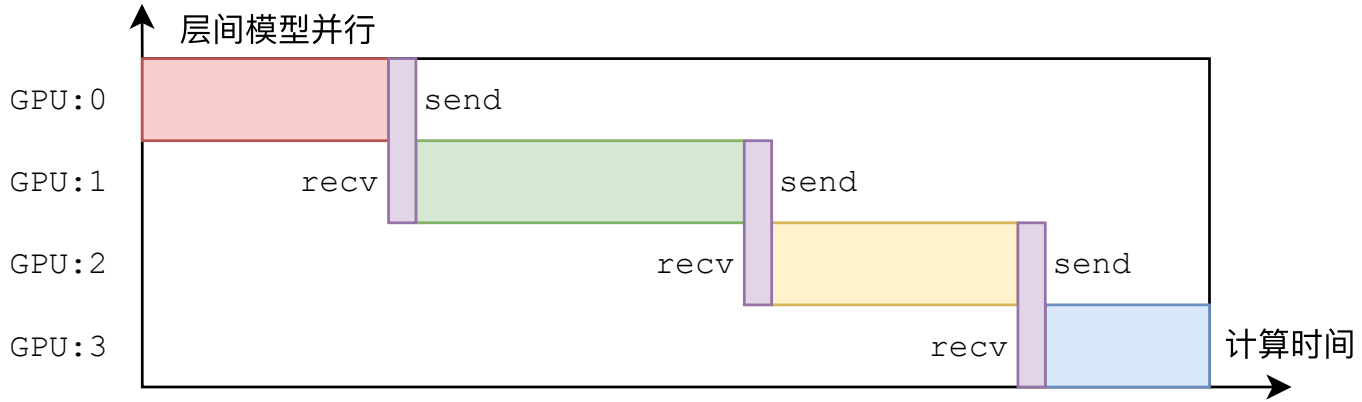
2. 层间模型并行



3. 层间模型并行(Inter-layer Model Parallelism)

MEGVII 旷视

```python
def send_to_next_gpu(tensor):
    shape, dtype = tensor.shape, np.dtype(tensor.dtype).name
    dist.get_client().user_set(f"shape_of_src{dist.get_rank()}", shape)
    dist.get_client().user_set(f"dtype_of_src{dist.get_rank()}", dtype)
    return F.distributed.remote_send(tensor, dest_rank=dist.get_rank() + 1)

def recv_fr_prev_gpu():
    shape = dist.get_client().user_get(f"shape_of_src{dist.get_rank() - 1}")
    shape = dist.get_client().user_get(f"dtype_of_src{dist.get_rank() - 1}")
    return F.distributed.remote_recv(src_rank=dist.get_rank() - 1, shape=shape, dtype=dtype)
```



3.1. 简单模型并行

普通数据并行

```
class ResNet18(M.Module):
    def __init__(self):
        super().__init__()
        self.stem = M.Sequential(
            M.ConvBn2d(3, 64, 7, stride=2, padding=3, bias=False),
            M.MaxPool2d(kernel_size=3, stride=2, padding=1),
        )
        self.features = M.Sequential(
            BasicBlock(64,  64,  1),
            BasicBlock(64,  64,  1),
            BasicBlock(64,  128, 2),
            BasicBlock(128, 128, 1),
            BasicBlock(128, 256, 2),
            BasicBlock(256, 256, 1),
            BasicBlock(256, 512, 2),
            BasicBlock(512, 512, 1),
        )
        self.classifier = M.Linear(512, 1000)

    def forward(self, x):
        x = self.stem(x)
        x = self.features(x)
        x = F.avg_pool2d(x, 7)
        x = F.flatten(x, 1)
        x = self.classifier(x)
        return x
```

简单模型并行

```
class ResNet18MP(M.Module):
    def __init__(self):
        super().__init__()
        self.classifier = None
        if dist.get_rank() == 0:
            self.features = M.Sequential(
                M.ConvBn2d(3, 64, 7, stride=2, padding=3, bias=False),
                M.MaxPool2d(kernel_size=3, stride=2, padding=1),
                BasicBlock(64,  64,  1),
                BasicBlock(64,  64,  1),
            )
        elif dist.get_rank() == 1:
            self.features = M.Sequential(
                BasicBlock(64,  128, 2),
                BasicBlock(128, 128, 1),
            )
        elif dist.get_rank() == 2:
            self.features = M.Sequential(
                BasicBlock(128, 256, 2),
                BasicBlock(256, 256, 1),
            )
        elif dist.get_rank() == 3:
            self.features = M.Sequential(
                BasicBlock(256, 512, 2),
                BasicBlock(512, 512, 1),
            )
            self.classifier = M.Linear(512, 1000)

    def forward(self, x):
        if dist.get_rank() > 0:
            x = recv_fr_prev_gpu()

        x = self.features(x)

        if dist.get_rank() != 3:
            _ = send_to_next_gpu(x)
        else:
            x = F.avg_pool2d(x, 7)
            x = F.flatten(x, 1)
            x = self.classifier(x)
        return x
```

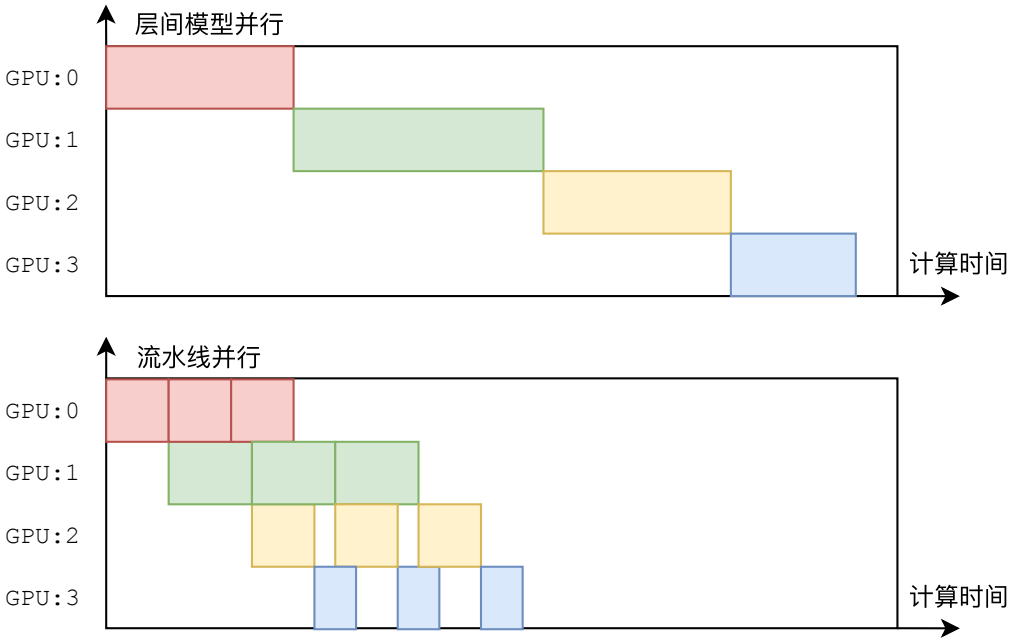
3.1. 简单模型并行（推理&训练）

```python
@dist.launcher(n_gpus=4)
def inference():
    m = ResNet18MP()
    x = F.ones([32, 3, 224, 224])
    y = m(x)
    print(y.shape)

# --- Get ---
# (32, 64, 56, 56)
# (32, 128, 28, 28)
# (32, 256, 14, 14)
# (32, 1000)
```

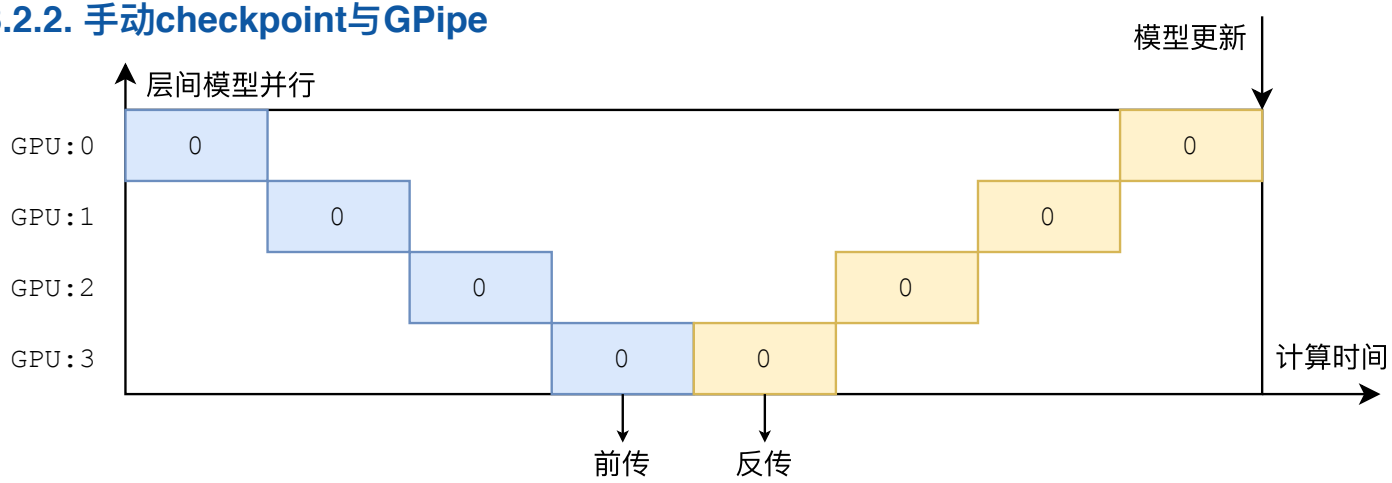
```python
@dist.launcher(n_gpus=4)
def train():
    m = ResNet18MP()
    x = F.ones([32, 3, 224, 224])
    label = F.zeros([32,], dtype="int32")

    gm = ad.GradManager().attach(m.parameters())
    opt = optim.SGD(m.parameters(), 1e-3, 0.9, 1e-4)

    for _ in range(2):
        with gm:
            y = m(x)
            if dist.get_rank() == 3:
                loss = F.nn.cross_entropy(y, label)
            else:
                loss = None
            gm.backward(loss)
        opt.step().clear_grad()
        print(loss)

# --- Get ---
# None
# None
# None
# Tensor(6.631501, device=gpu3:0)
# None
# None
# None
# Tensor(5.7957726, device=gpu3:0)
```

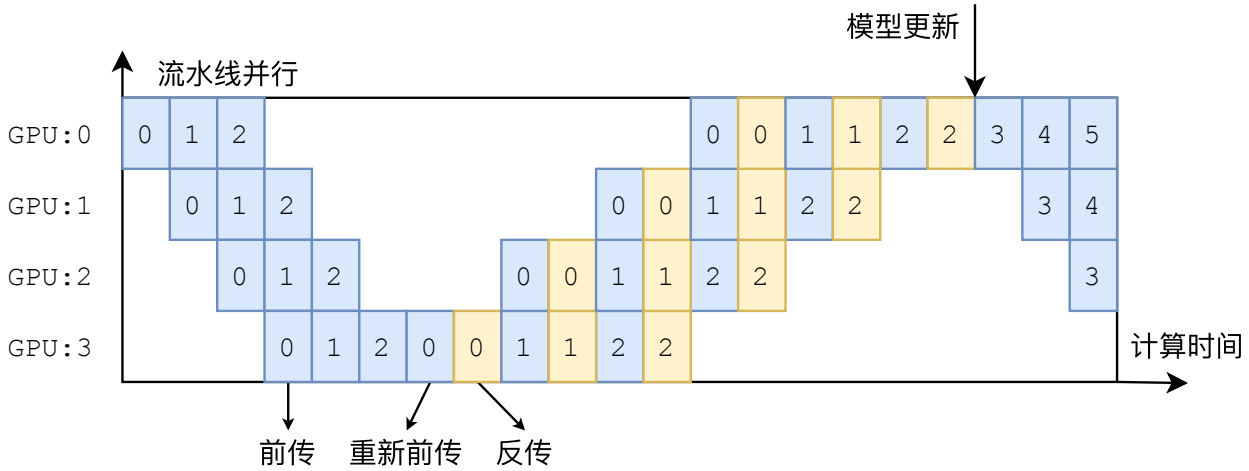
3.2. 流水线并行(Pipeline Parallelism)

```
def forward(self, x):
    self.num_chunks = 4
    self.inp_chunks = []
    self.oup_chunks = []
    if dist.get_rank() == 0:
        self.inp_chunks = F.split(x, 4)

    for i in range(self.num_chunks):
        if dist.get_rank() == 0:
            x = self.inp_chunks[i]
        else:
            x = recv_fr_prev_gpu()
            self.inp_chunks.append(x)

        x = self.features(x)
        if dist.get_rank() != 3:
            _ = send_to_next_gpu(x)
        else:
            x = F.avg_pool2d(x, 7)
            x = F.flatten(x, 1)
            x = self.classifier(x)
        self.oup_chunks.append(x)

    return F.concat(self.oup_chunks)
```



3.2.2. 手动checkpoint与GPipe





3.2.2. 手动checkpoint与GPipe

MEGVII 旷视

前传（不计算梯度）

```
def forward(self, x):
    self.num_chunks = 4
    self.inp_chunks = []
    self.oup_chunks = []
    if dist.get_rank() == 0:
        self.inp_chunks = F.split(x, 4)

    for i in range(self.num_chunks):
        if dist.get_rank() == 0:
            x = self.inp_chunks[i]
        else:
            x = recv_fr_prev_gpu()
            self.inp_chunks.append(x)

        x = self.features(x)
        if dist.get_rank() != 3:
            _ = send_to_next_gpu(x)
        else:
            x = F.avg_pool2d(x, 7)
            x = F.flatten(x, 1)
            x = self.classifier(x)
        self.oup_chunks.append(x)

    return F.concat(self.oup_chunks)
```

反传（重新计算一遍前传）

```
def backward(self, label, gm):
    label_chunks = F.split(label, 4)
    losses = []

    for i, x in enumerate(self.inp_chunks):
        with gm:
            gm.attach(x)  # query gradient of the input
            y = self.features(x)

            if dist.get_rank() == 3:
                y = F.avg_pool2d(y, 7)
                y = F.flatten(y, 1)
                y = self.classifier(y)
                loss = F.nn.cross_entropy(y, label_chunks[i])
                losses.append(loss)
                gm.backward(loss)
            else:
                grad = grad_fr_next_gpu()
                gm.backward(y, dy=grad)

            if dist.get_rank() != 0:
                _ = grad_to_prev_gpu(x.grad)

    return sum(losses) / self.num_chunks if losses else None
```

Code Available:
https://github.com/zhouyizhuang-megvii/MegEngineParallelTutorial