大模型：分布式训练

PTD并行配置

ZOMI

大模型业务全流程

数据 & 模型算法

2. 数据处理

- 开源数据
- 数据预处理
- 向量数据库

3. 模型算法

- LLM模型架构
- 多模态一切皆Tokens

模型训练 & 微调

4. 模型训练

- 混合精度
- 梯度检查
- 梯度累积
- …
- 5. 分布式并行
- 训练集群稳定性

6. 模型微调

- 全参微调
- 低参微调
- 指令微调

模型验证 & 推理部署

7. 模型验证

- 下游任务
- 测评标准
- bench mark

8. 推理与智能体

- 量化压缩
- 推理加速
- 9. Agent 智能体

集群算力准备

1. AI 集群建设

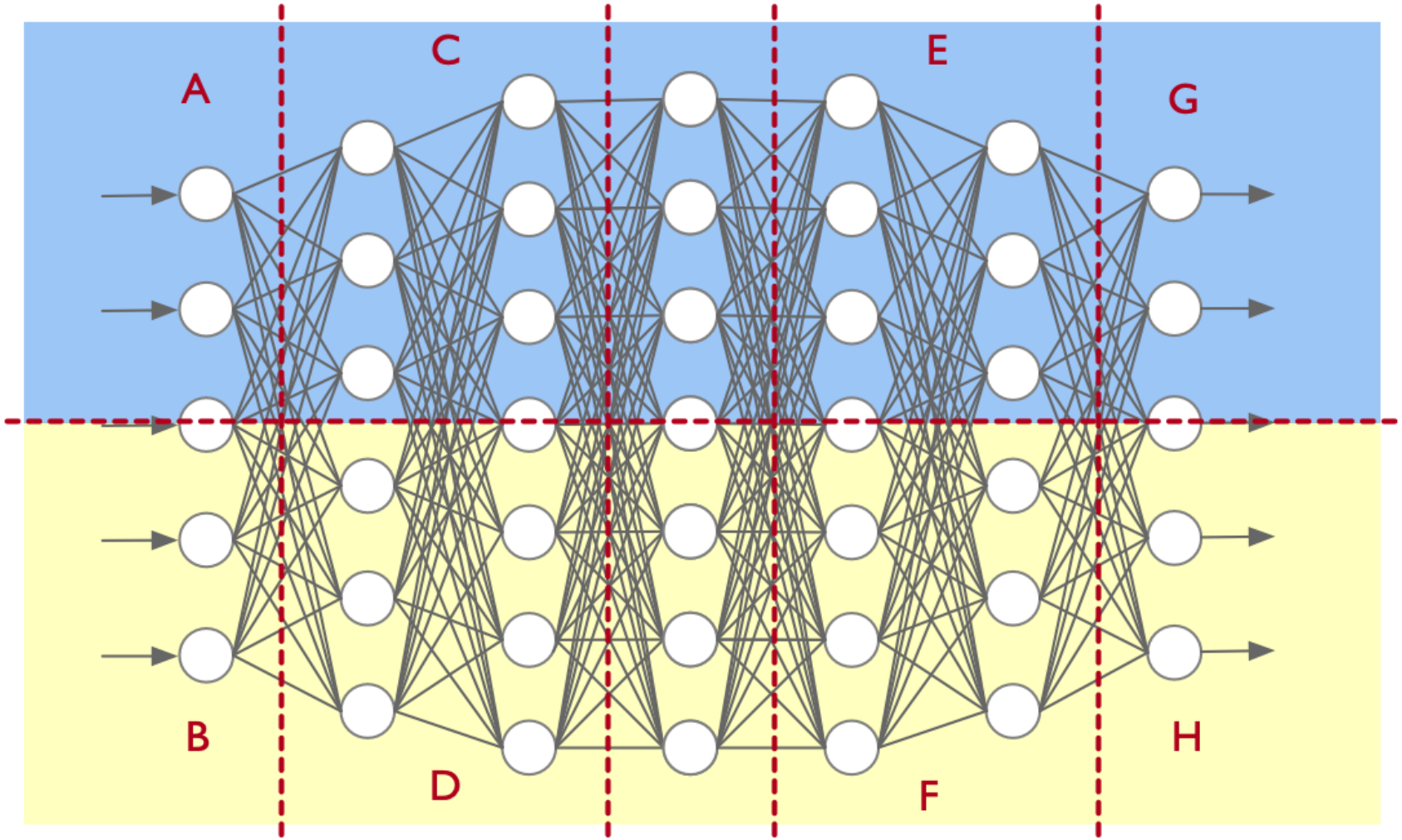
- 计算、存储、网络
- AI 集群机房建设
- AI 集群上线与运维

大模型系列 – 分布式训练加速

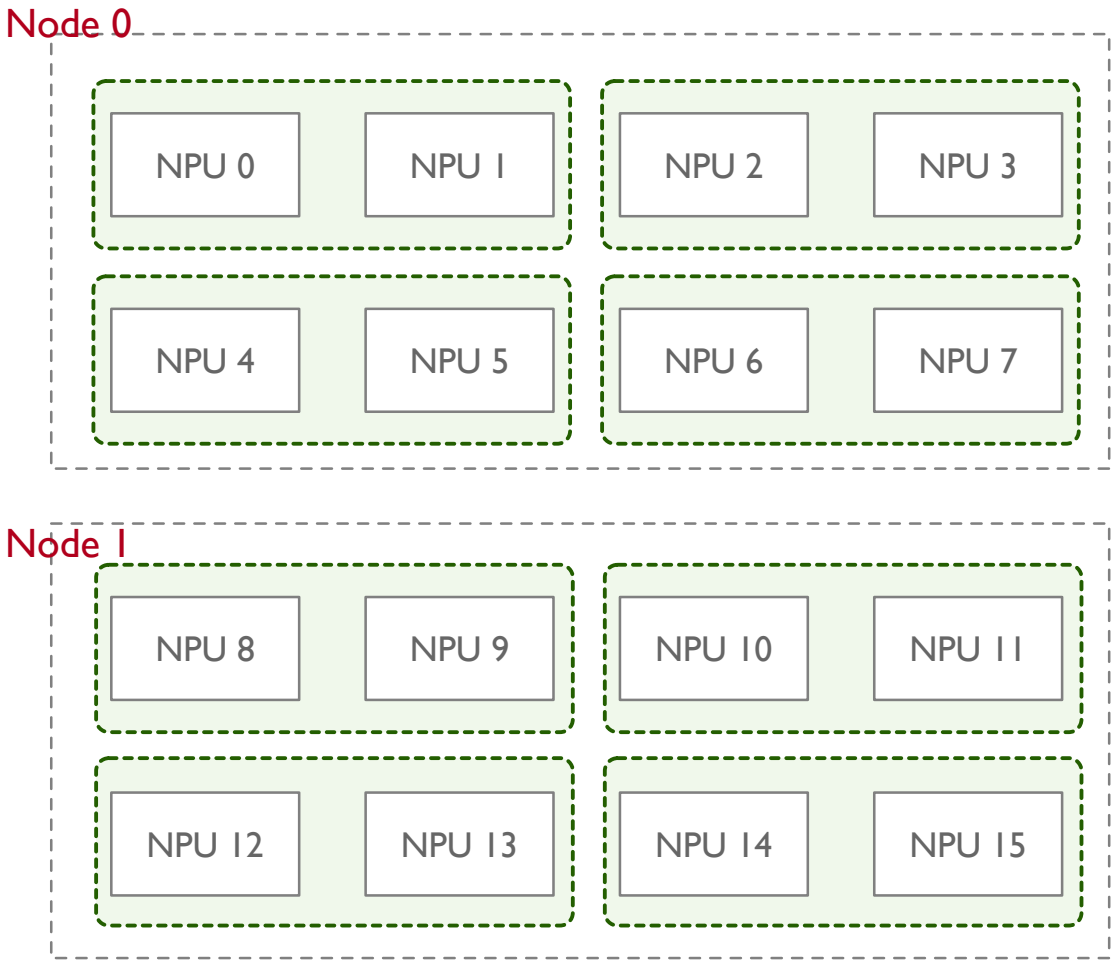
- **具体内容**

1. **分布式加速库：**
   - 业界常用分布式加速库 & 作用
2. **DeepSpeed 特性：**
   - 基本概念 - 整体框架 – Zero-1/2/3 – ZeRO-Offload – ZeRO-Infinity
3. **Megatron 特性：**
   1. 总体介绍 – 整体流程 – 并行配置 – DP – TP – PP

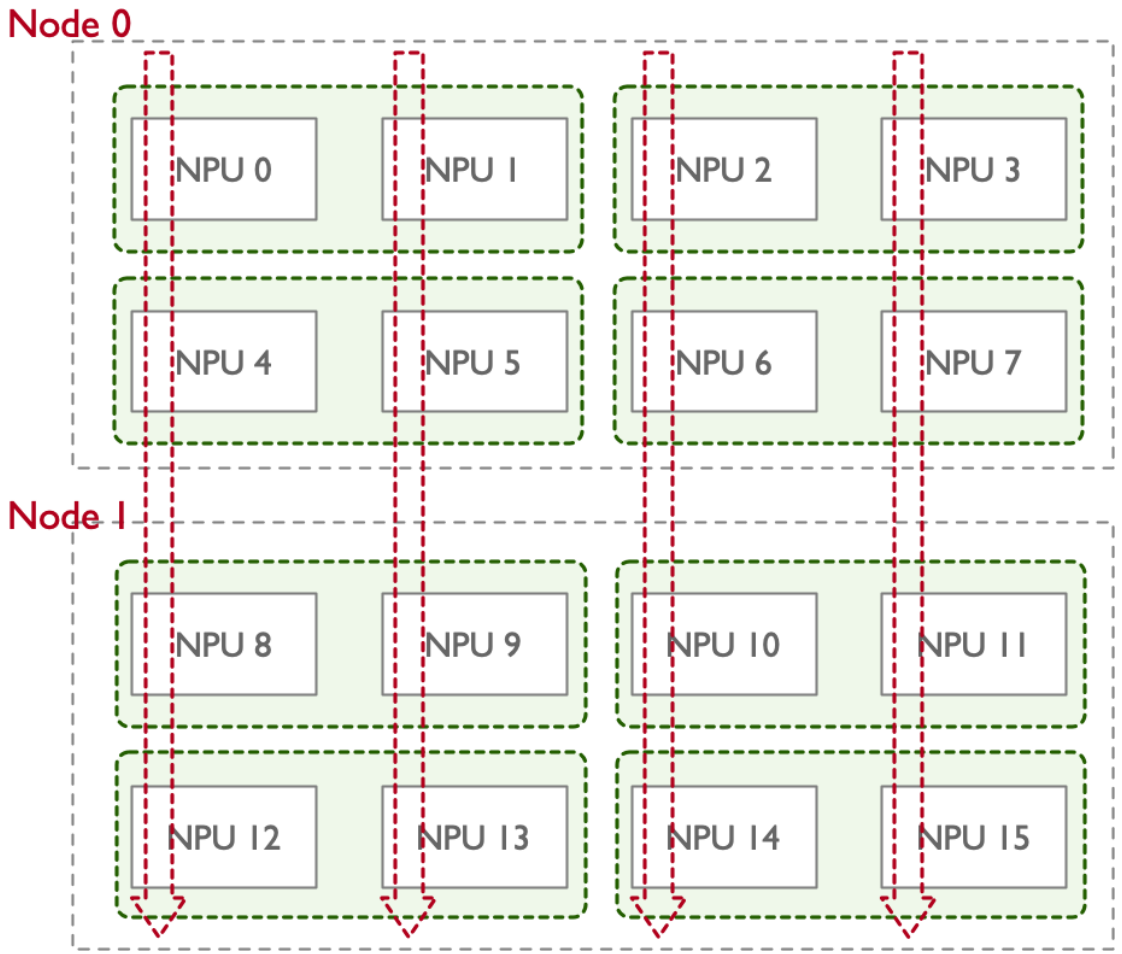
PTD 并行在集群里面与模型的关系



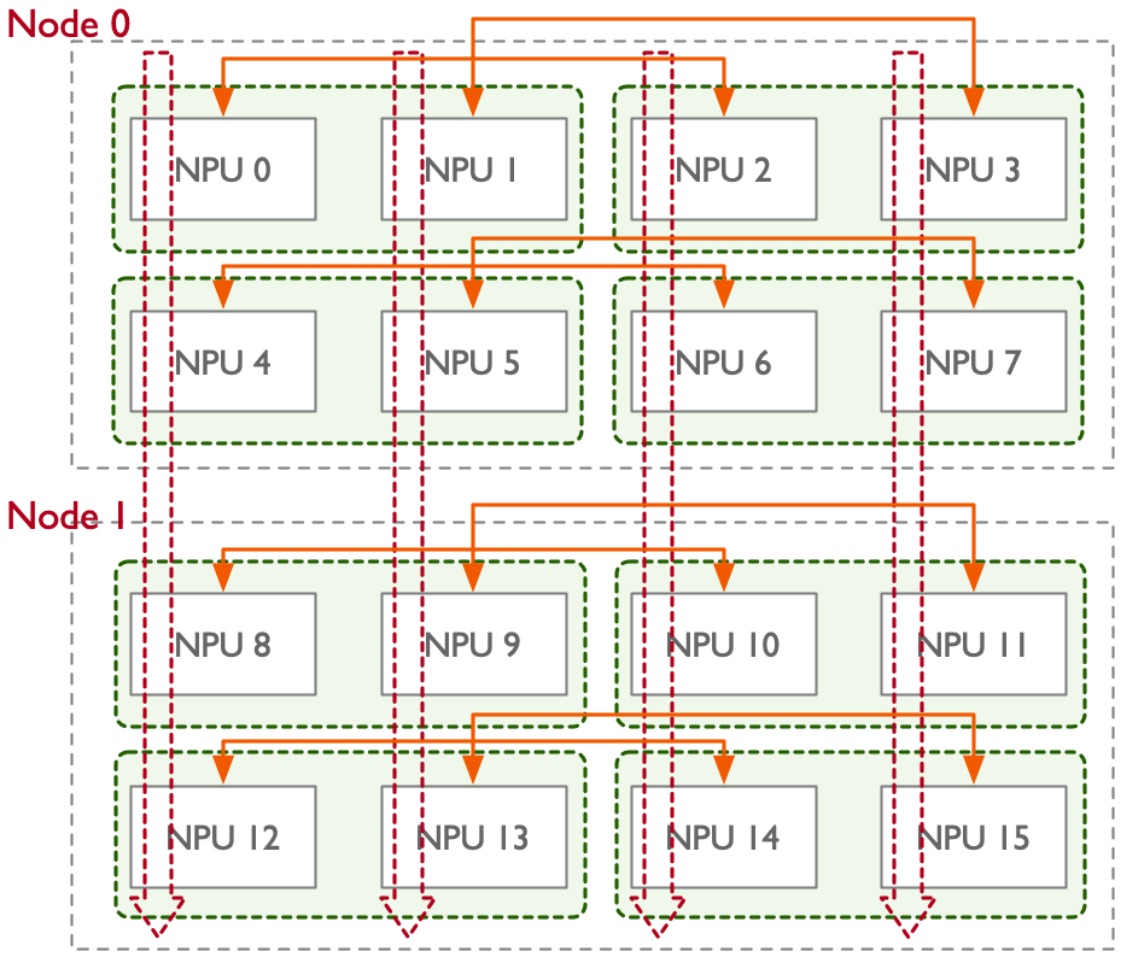
- Tensor Parallel



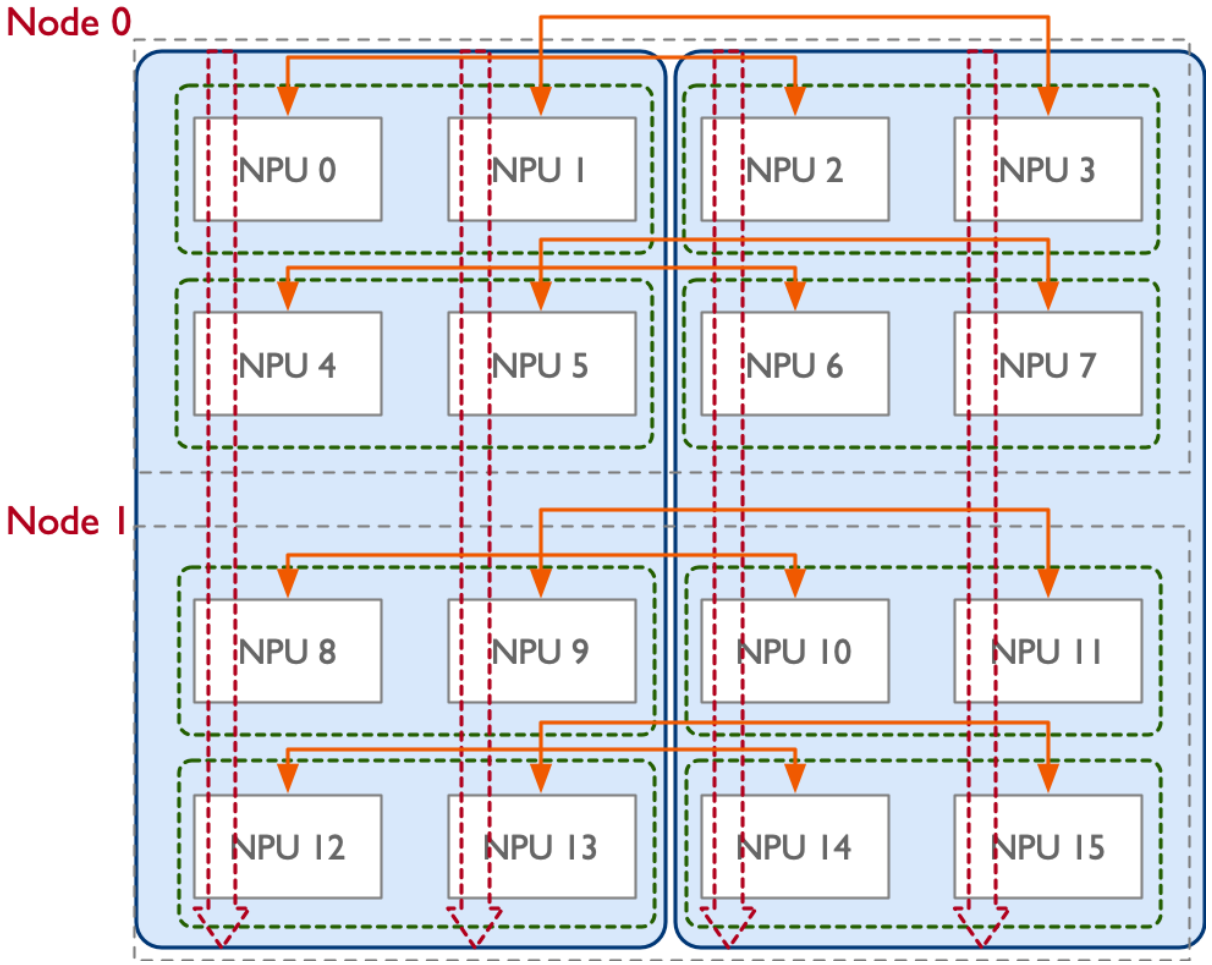
- Pipeline Parallel



- Data Parallel



- Data Parallel



1. Megatron-LM

PTD回顾与初始化

initialize_model_parallel() 并行配置初始化

```
12  # Intra-layer model parallel group that the current rank belongs to.
13  _TENSOR_MODEL_PARALLEL_GROUP = None
14  # Inter-layer model parallel group that the current rank belongs to.
15  _PIPELINE_MODEL_PARALLEL_GROUP = None
16  # Model parallel group (both intra- and pipeline) that the current ra
17  _MODEL_PARALLEL_GROUP = None
18  # Embedding group.
19  _EMBEDDING_GROUP = None
20  # Position embedding group.
21  _POSITION_EMBEDDING_GROUP = None
22  # Data parallel group that the current rank belongs to.
23  _DATA_PARALLEL_GROUP = None
24  _DATA_PARALLEL_GROUP_GLOO = None
25  # tensor model parallel group and data parallel group combined
26  # used for fp8 and moe training
27  _TENSOR_AND_DATA_PARALLEL_GROUP = None
28  # Expert parallel group that the current rank belongs to.
29  _TENSOR_AND_EXPERT_PARALLEL_GROUP = None
30  _DATA_MODULO_EXPERT_PARALLEL_GROUP = None
31  _DATA_MODULO_EXPERT_PARALLEL_GROUP_GLOO = None
32
33
34  _VIRTUAL_PIPELINE_MODEL_PARALLEL_RANK = None
35  _VIRTUAL_PIPELINE_MODEL_PARALLEL_WORLD_SIZE = None
36  _PIPELINE_MODEL_PARALLEL_SPLIT_RANK = None
37
38  # These values enable us to change the mpu sizes on the fly.
39  _MPU_TENSOR_MODEL_PARALLEL_WORLD_SIZE = None
40  _MPU_PIPELINE_MODEL_PARALLEL_WORLD_SIZE = None
41  _MPU_EXPERT_MODEL_PARALLEL_WORLD_SIZE = None
```

negatron > core > parallel_state.py > initialize_model_parallel

模型进行分组，初始化进程组相关的全局变量

```
97  def initialize_model_parallel(
172             GPUs of context parallelism on data parallel group for
173             weight gradient all-reduce.
174
175         nccl_communicator_config_path (str, default = None):
176             Path to the yaml file of NCCL communicator configurations.
177             `min_ctas`, `max_ctas`, and `cga_cluster_size` can be set
178             for each communicator.
179
180     Let's say we have a total of 16 GPUs denoted by g0 ... g15 and we
181     use 2 GPUs to parallelize the model tensor, and 4 GPUs to parallelize
182     the model pipeline. The present function will
183     create 8 tensor model-parallel groups, 4 pipeline model-parallel groups
184     and 8 data-parallel groups as:
185         8 data_parallel groups:
186             [g0, g2], [g1, g3], [g4, g6], [g5, g7], [g8, g10], [g9, g11], [g12, g14]
187         8 tensor model-parallel groups:
188             [g0, g1], [g2, g3], [g4, g5], [g6, g7], [g8, g9], [g10, g11], [g12, g13]
189         4 pipeline model-parallel groups:
190             [g0, g4, g8, g12], [g1, g5, g9, g13], [g2, g6, g10, g14], [g3, g7, g11,
191     Note that for efficiency, the caller should make sure adjacent ranks
192     are on the same DGX box. For example if we are using 2 DGX-1 boxes
193     with a total of 16 GPUs, rank 0 to 7 belong to the first box and
194     ranks 8 to 15 belong to the second box.
195
196     """
197     # Get world size and rank. Ensure some consistencies.
198     assert torch.distributed.is_initialized()
199     world_size: int = torch.distributed.get_world_size()
200
201     if (
202         world_size
203         % (tensor_model_parallel_size * pipeline_model_parallel_size * context_paral
```

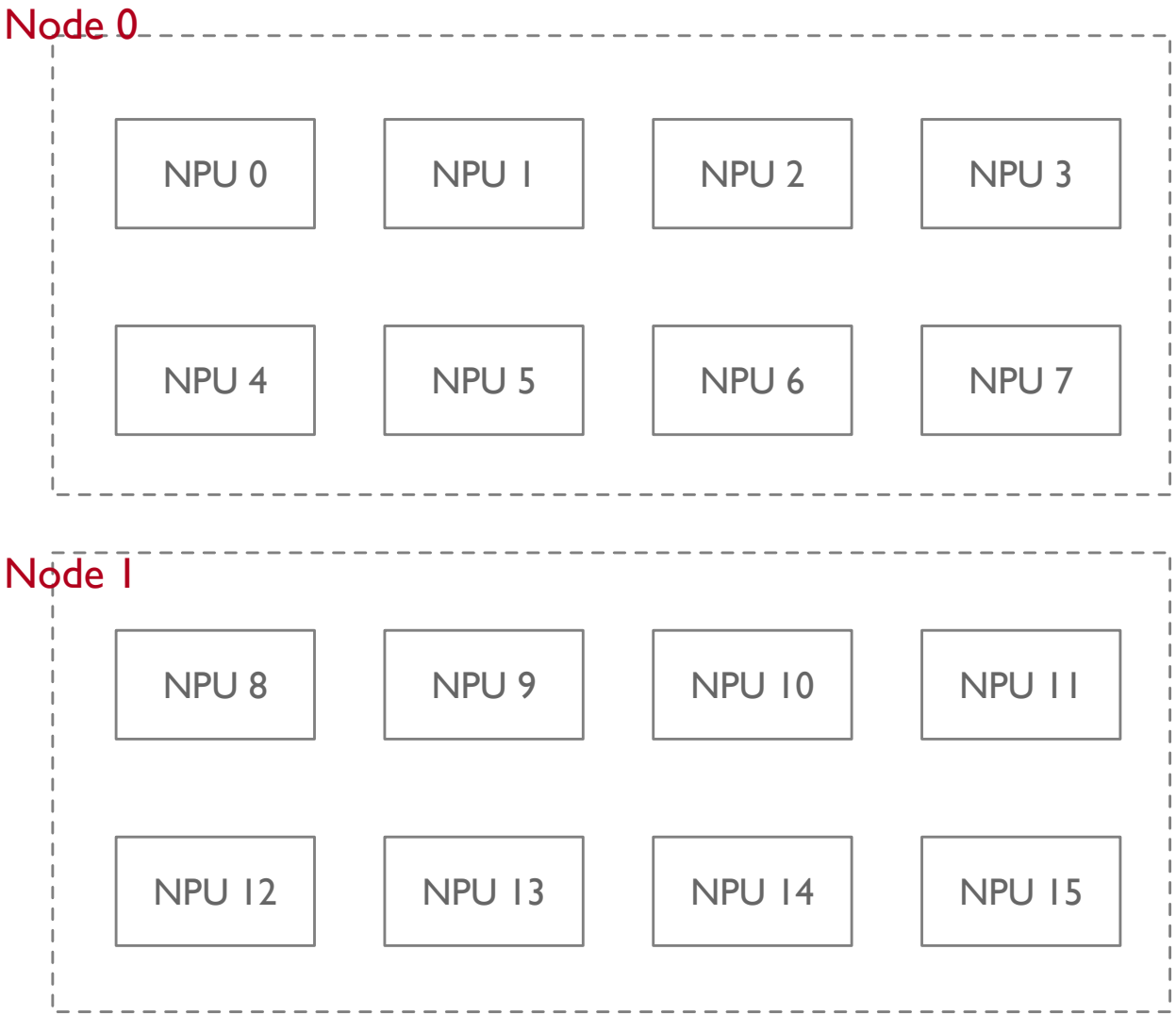
initialize_model_parallel() 并行配置初始化

模型进行分组，初始化进程组相关的全局变量

```
negatron > core > parallel_state.py > initialize_model_parallel
97  def initialize_model_parallel(
172             GPUs of context parallelism on data parallel group for
173             weight gradient all-reduce.
174
175         nccl_communicator_config_path (str, default = None):
176             Path to the yaml file of NCCL communicator configurations.
177             `min_ctas`, `max_ctas`, and `cga_cluster_size` can be set
178             for each communicator.
179
180     Let's say we have a total of 16 GPUs denoted by g0 ... g15 and we
181     use 2 GPUs to parallelize the model tensor, and 4 GPUs to parallelize
182     the model pipeline. The present function will
183     create 8 tensor model-parallel groups, 4 pipeline model-parallel groups
184     and 8 data-parallel groups as:
185         8 data_parallel groups:
186             [g0, g2], [g1, g3], [g4, g6], [g5, g7], [g8, g10], [g9, g11], [g12, g14]
187         8 tensor model-parallel groups:
188             [g0, g1], [g2, g3], [g4, g5], [g6, g7], [g8, g9], [g10, g11], [g12, g13]
189         4 pipeline model-parallel groups:
190             [g0, g4, g8, g12], [g1, g5, g9, g13], [g2, g6, g10, g14], [g3, g7, g11,
191     Note that for efficiency, the caller should make sure adjacent ranks
192     are on the same DGX box. For example if we are using 2 DGX-1 boxes
193     with a total of 16 GPUs, rank 0 to 7 belong to the first box and
194     ranks 8 to 15 belong to the second box.
195
196     """
197     # Get world size and rank. Ensure some consistencies.
198     assert torch.distributed.is_initialized()
199     world_size: int = torch.distributed.get_world_size()
200
201     if (
202         world_size
203         % (tensor_model_parallel_size * pipeline_model_parallel_size * context_para
```

- 假定16 NPU，两个 node，rank 0~7 第一个节点，rank 8~15 属于第二个节点：
  1. TP 组大小 2，16 个 NPU 被分成 8 组： [g0, g1], [g2, g3], [g4, g5], [g6, g7], [g8, g9], [g10, g11], [g12, g13], [g14, g15]
  2. PP 组大小 4，16 个 NPU 被分成 4 组： [g0, g4, g8, g12], [g1, g5, g9, g13], [g2, g6, g10, g14], [g3, g7, g11, g15]
  3. DP 组大小 2，16 个 NPU 被分成 8 组：[g0, g2], [g1, g3], [g4, g6], [g5, g7], [g8, g10], [g9, g11], [g12, g14], [g13, g15]

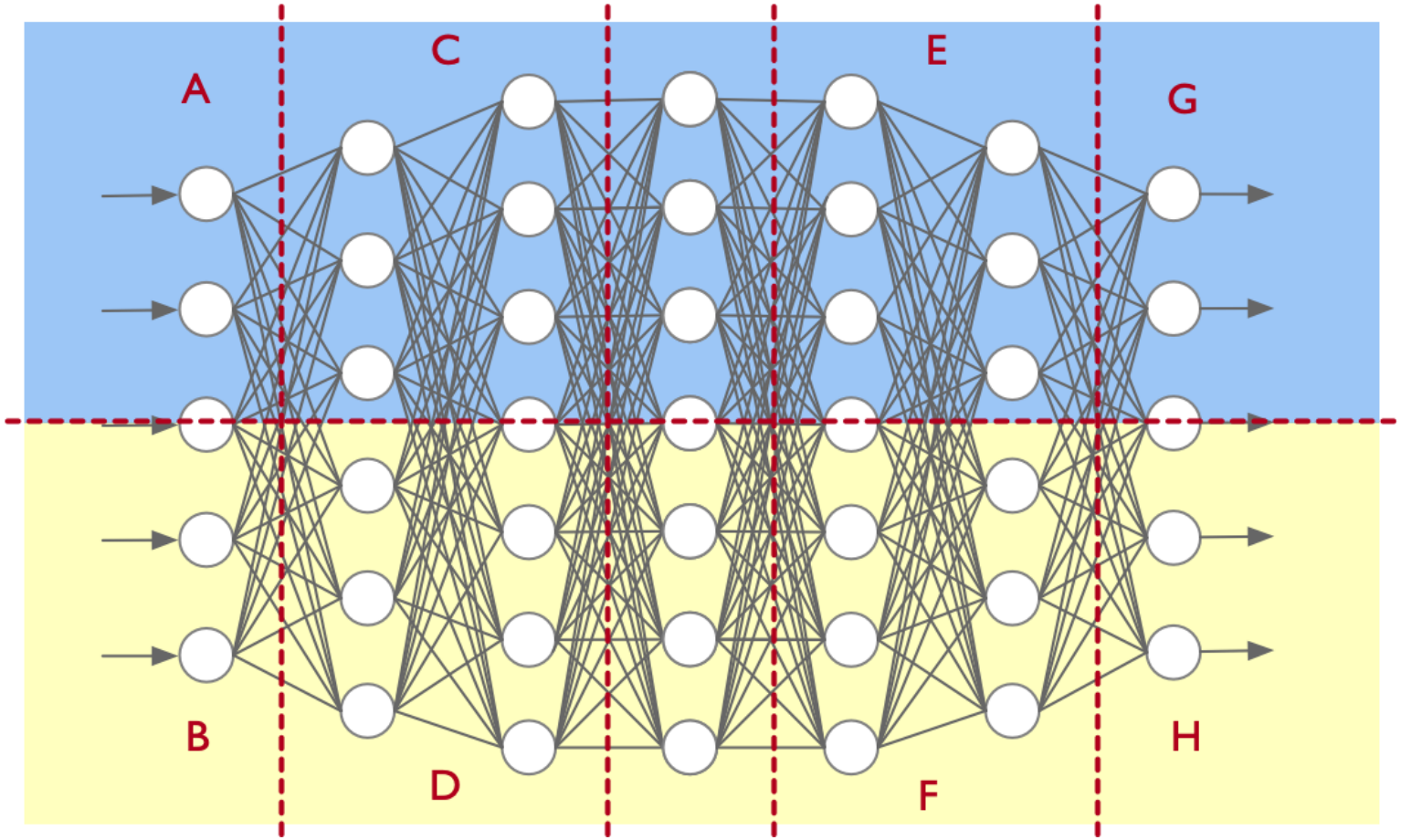
并行配置 & 节点情况



- Node 1 : NPU 0 ~ 7
- Node 2 : NPU 8 ~ 15

- world size = 16
- tensor model parallel size = 2
- pipeline model parallel size = 4
- data parallel size = 2

模型并行切分配置



- world size = 16
- tensor model parallel size = 2
- pipeline model parallel size = 4
- 模型并行（TP+PP）：
  - TP 组大小 2，16 个 NPU 被分成 8 组：[g0, g1], [g2, g3], [g4, g5], [g6, g7], [g8, g9], [g10, g11], [g12, g13], [g14, g15]
  - PP 组大小 4，16 个 NPU 被分成 4 组：[g0, g4, g8, g12], [g1, g5, g9, g13], [g2, g6, g10, g14], [g3, g7, g11, g15]

并行配置 & 处理进程 process groups

```
world_size = 16
tensor_model_parallel_size = 2
pipeline_model_parallel_size = 4
data_parallel_size = world_size // (tensor_model_parallel_size *
                                    pipeline_model_parallel_size) # 2
num_tensor_model_parallel_groups = world_size // tensor_model_parallel_size # 8
num_pipeline_model_parallel_groups = world_size // pipeline_model_parallel_size # 4
num_data_parallel_groups = world_size // data_parallel_size # 8

# Build the data-parallel groups.
print("Build DP Groups :")
all_data_parallel_group_ranks = []
for i in range(pipeline_model_parallel_size):
    start_rank = i * num_pipeline_model_parallel_groups
    end_rank = (i + 1) * num_pipeline_model_parallel_groups
    for j in range(tensor_model_parallel_size):
        ranks = range(start_rank + j, end_rank,
                      tensor_model_parallel_size)
        all_data_parallel_group_ranks.append(list(ranks))
print(all_data_parallel_group_ranks)
```

从 TP 看，分成 8 个进程 group

从 PP 看，分成 4 个 进程 group

从 DP 看分成 8 个 进程 group

即有 8 个 DDP，每个 DDP 包括 2 个 rank

2 Megatron-LM

TP 张量并行

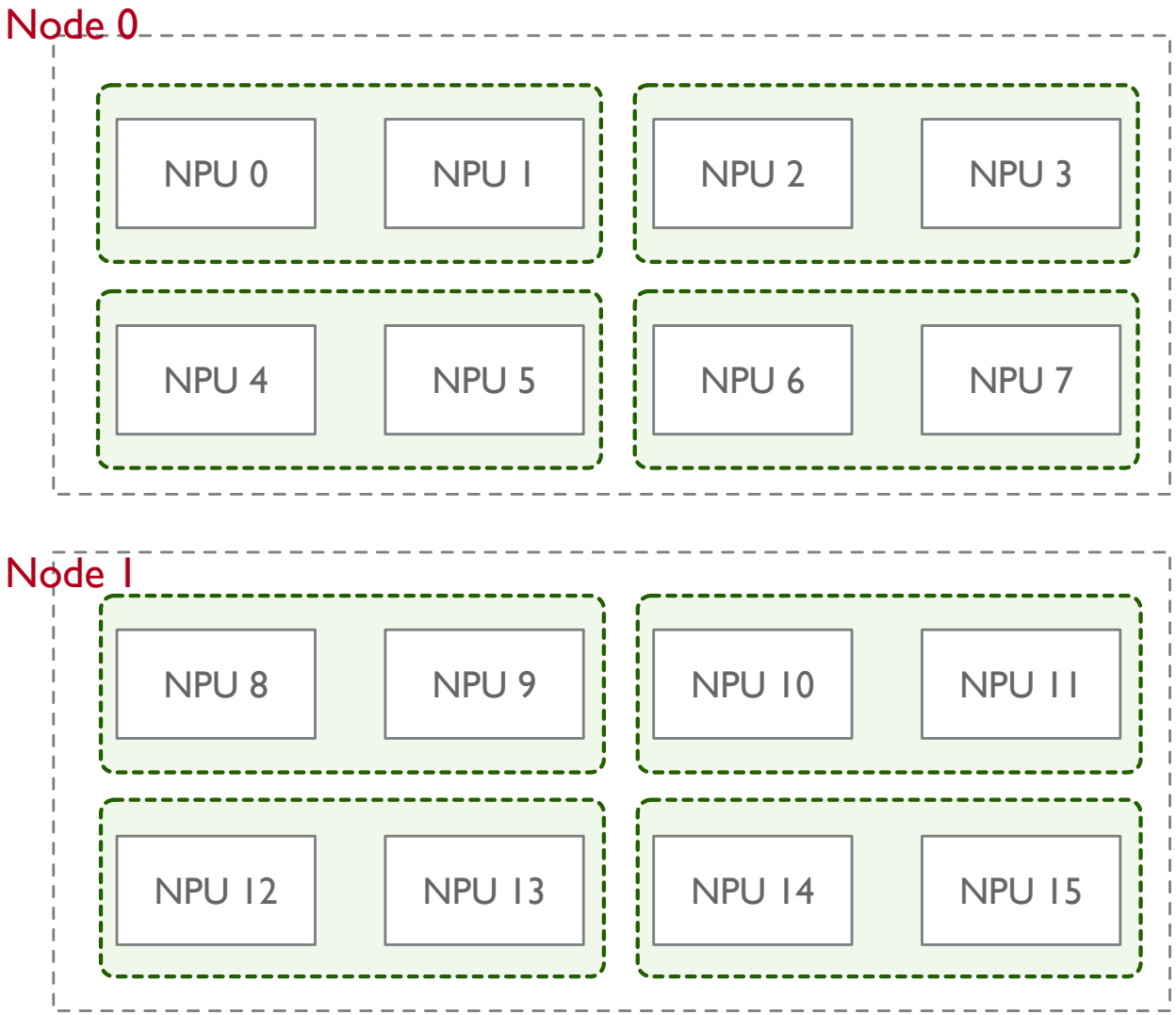
TP 进程组分配

```
350
351     # Build the tensor model-parallel groups.
352     global _TENSOR_MODEL_PARALLEL_GROUP
353     assert (
354         _TENSOR_MODEL_PARALLEL_GROUP is None
355     ), 'tensor model parallel group is already initialized'
356     for i in range(num_tensor_model_parallel_groups):
357         ranks = range(i * tensor_model_parallel_size,
358                       (i + 1) * tensor_model_parallel_size)
359         group = torch.distributed.new_group(
360             ranks, pg_options=get_nccl_options('tp', nccl_comm_cfgs)
361         )
362         if rank in ranks:
363             _TENSOR_MODEL_PARALLEL_GROUP = group
364
```

每一个 tensor-model-parallel group 的 rank 一定相邻

记录了当前 rank 的进程组信息
e.g. ， rank2 = group([g2, g3])

TP 进程组 rank 分配



- world size = 16 ， TP = 2
- group 16 / 2 = 8
  - 分为 8 个进程组 group，每个进程组有两个 rank
  - [g0, g1] 为大模型某一层切分为 2 份，分别被 rank g0, g1 执行
  - [g2, g3] 为大模型另一层切分为2份，分别被 rank g2，g3 执行
  - group 中进行通信主要用到 reduce()、gather()、split()

TP 进程组使用

```
520
521   def get_tensor_model_parallel_group(check_initialized=True):
522       """Get the tensor model parallel group the caller rank belongs to."""
523       if check_initialized:
524           assert (
525               _TENSOR_MODEL_PARALLEL_GROUP is not None
526           ), 'tensor model parallel group is not initialized'
527       return _TENSOR_MODEL_PARALLEL_GROUP
528
```

返回自身 rank 对应的 TP 进程组 group

```
14
15  def _reduce(input_):
16      """All-reduce the input tensor across model parallel group."""
17
18      # Bypass the function if we are using only 1 GPU.
19      if get_tensor_model_parallel_world_size() == 1:
20          return input_
21
22      # All-reduce.
23      torch.distributed.all_reduce(input_, group=get_tensor_model_parallel_group())
24
25      return input_
26
```

当 TP 反向传播时，利用 _TENSOR_MODEL_PARALLEL_GROUP group 组内进行集合通信

3 Megatron-LM

PP 模型并行

PP 进程组分配

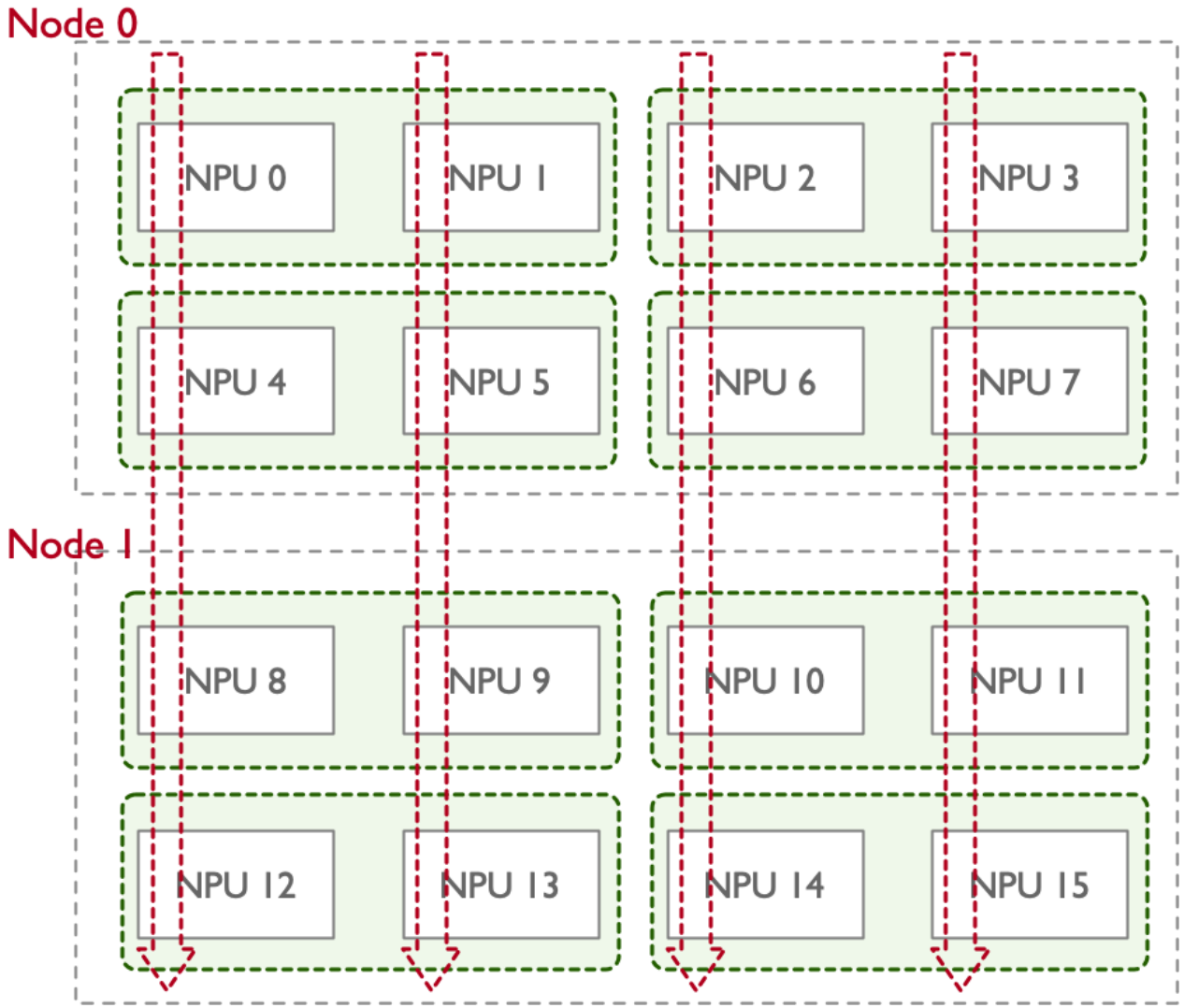
```
370         _PIPELINE_MODEL_PARALLEL_GROUP is None
371     ), 'pipeline model parallel group is already initialized'
372     global _EMBEDDING_GROUP
373     global _EMBEDDING_GLOBAL_RANKS
374     assert _EMBEDDING_GROUP is None, 'embedding group is already initialized'
375     global _POSITION_EMBEDDING_GROUP
376     global _POSITION_EMBEDDING_GLOBAL_RANKS
377     assert _POSITION_EMBEDDING_GROUP is None, 'position embedding group is already initialized'
378     for i in range(num_pipeline_model_parallel_groups):
379         ranks = range(i, world_size, num_pipeline_model_parallel_groups)
380         group = torch.distributed.new_group(
381             ranks, pg_options=get_nccl_options('pp', nccl_comm_cfgs)
382         )
383         if rank in ranks:
384             _PIPELINE_MODEL_PARALLEL_GROUP = group
385             _PIPELINE_GLOBAL_RANKS = ranks
386         # Setup embedding group (to exchange
387         # first and last stages).
388         if len(ranks) > 1:
```

rank 每隔 n // p个取一个

当前 rank 对应的 PP 进程组

当前进程组的 ranks
e.g.,rank2 = [g2, g6, g10, g14]

PP 进程组 rank 分配



- world size = 16 , PP = 4
- group16 / 4 = 4
  - 流水线 group 每隔 n // p 个取一个，stage i 的 rank 范围是：[(i-1) * n//p, (i) * n//p]
  - 分为 4 个进程组 group，每个进程组有 4 个 rank，每个 rank 执行 Pipeline 一个 Stage
  - Pipeline Stage 为 4， 串行执行。即 group1 [g0, g4, g8, g12] 4 个 NPU 串行。

PP 进程组使用

```python
def _batched_p2p_ops(
    *,
    tensor_send_prev: Optional[torch.Tensor],
    tensor_recv_prev: Optional[torch.Tensor],
    tensor_send_next: Optional[torch.Tensor],
    tensor_recv_next: Optional[torch.Tensor],
    group: torch.distributed.ProcessGroup
):
    ops = []
    if tensor_send_prev is not None:
        send_prev_op = torch.distributed.P2POp(
            torch.distributed.isend,
            tensor_send_prev,
            get_pipeline_model_parallel_prev_rank(),
            group,
        )
        ops.append(send_prev_op)
    if tensor_recv_prev is not None:
        recv_prev_op = torch.distributed.P2POp(
            torch.distributed.irecv,
```

当 TP 进行通信时，使用 P2POp 点对点的通信方式，而不是集合通信方式

PP 进程组使用

```
529
530     def get_pipeline_model_parallel_group():
531         """Get the pipeline model parallel group the caller rank belongs to."""
532         assert (
533             _PIPELINE_MODEL_parallel_GROUP is not None
534         ), 'pipeline_model parallel group is not initialized'
535         return _PIPELINE_MODEL_PARALLEL_GROUP
536
```

返回自身 rank 对应的 PP 进程组 group

```
862
863     def get_pipeline_model_parallel_next_rank():
864         """Return the global rank that follows the caller in the pipeline"""
865         assert _PIPELINE_GLOBAL_RANKS is not None, "Pipeline parallel group is not initialized"
866         rank_in_pipeline = get_pipeline_model_parallel_rank()
867         world_size = get_pipeline_model_parallel_world_size()
868         return _PIPELINE_GLOBAL_RANKS[(rank_in_pipeline + 1) % world_size]
869
```

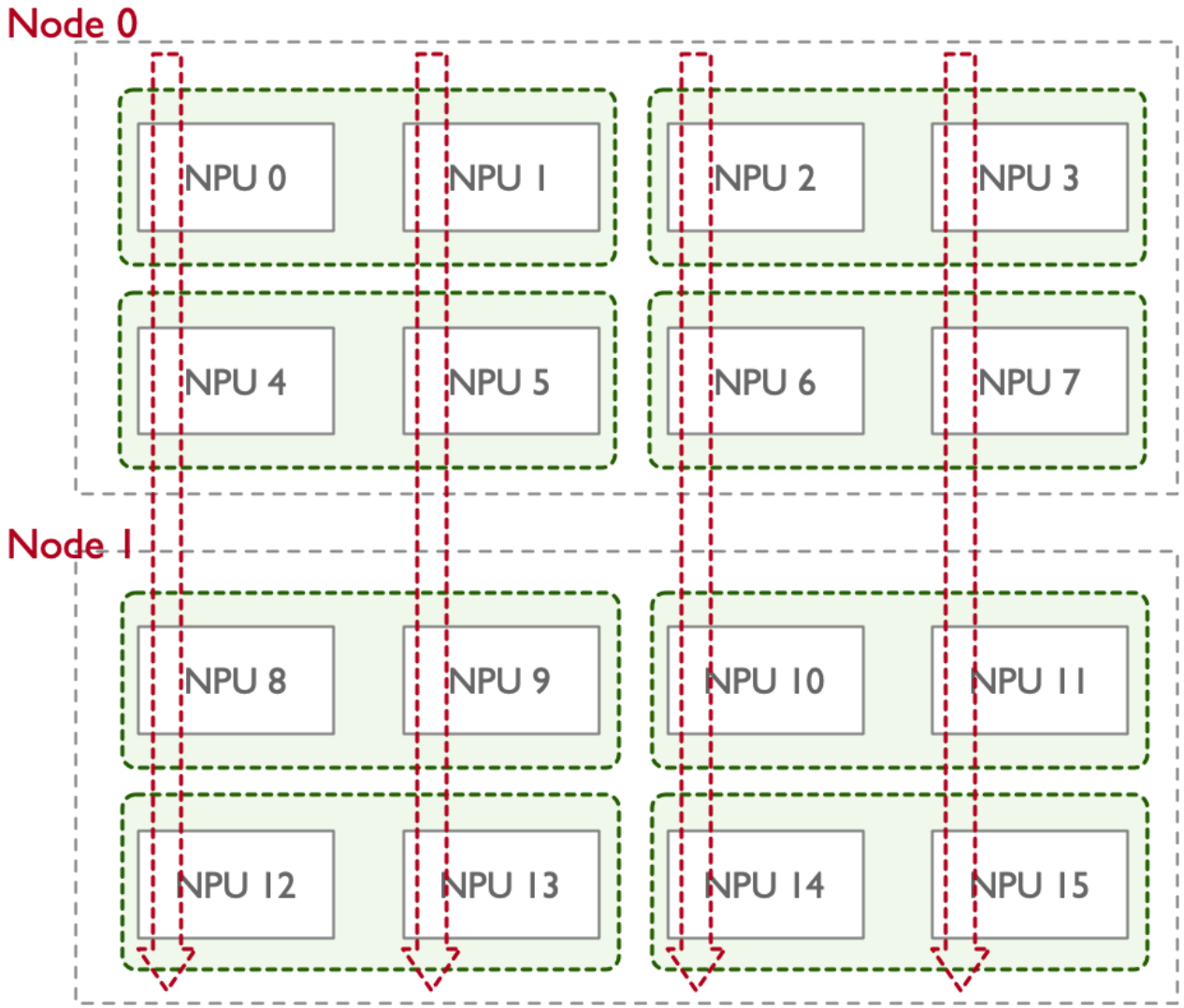
通过 _PIPELINE_GLOBAL_RANKS 得到了进程组内 ranks 信息

```
855     def get_pipeline_model_parallel_last_rank():
856         """Return the global rank of the last process in the pipeline for the
857         current tensor parallel group"""
858         assert _PIPELINE_GLOBAL_RANKS is not None, "Pipeline parallel group is not initialized"
859         last_rank_local = get_pipeline_model_parallel_world_size() - 1
860         return _PIPELINE_GLOBAL_RANKS[last_rank_local]
```

4 Megatron-LM

DP 数据并行

DP 进程组 rank 分配



- TxP 即放下一个大模型所需要 NPU 资源数：
  - d = (总 NPU / 一个模型需要资源) = n / (t * p)
  - n 个 NPU 可以同时训练 d 个大模型，可用 d 个 mini-batches 输入到 d 个大模型中进行训练
  - 因此 → 数据并行 DP 维度为 d

DP 进程组分配

```
264     global _DATA_PARALLEL_GLOBAL_RANKS_WITH_CP
265     assert _DATA_PARALLEL_GROUP is None, 'data parallel group is already initialized'
266     all_data_parallel_group_ranks_with_cp = []
267     for i in range(pipeline_model_parallel_size):
268         start_rank = i * num_pipeline_model_parallel_groups
269         end_rank = (i + 1) * num_pipeline_model_parallel_groups
270         for j in range(context_parallel_size * tensor_model_parallel_size):
271             ranks = range(
272                 start_rank + j, end_rank, context_parallel_size * tensor_model_parallel_size
273             )
274             group = torch.distributed.new_group(
275                 ranks, pg_options=get_nccl_options('dp', nccl_comm_cfgs)
276             )
277             group_gloo = torch.distributed.new_group(ranks, backend="gloo")
278             if rank in ranks:
279                 _DATA_PARALLEL_GROUP = group
280                 _DATA_PARALLEL_GROUP_GLOO = group_gloo
281                 _DATA_PARALLEL_GLOBAL_RANKS = ranks
282         for j in range(tensor_model_parallel_size):
283             ranks_with_cp = range(start_rank + j, end_rank, tensor_model_parallel_size)
284             all_data_parallel_group_ranks_with_cp.append(list(ranks_with_cp))
285             group_with_cp = torch.distributed.new_group(
286                 ranks_with_cp, pg_options=get_nccl_options('dp_cp', nccl_comm_cfgs)
287             )
```

1. 遍历流水线深度

2. 找到每个 stage 起始 start rank

3. 找到每个 stage 终止 end rank

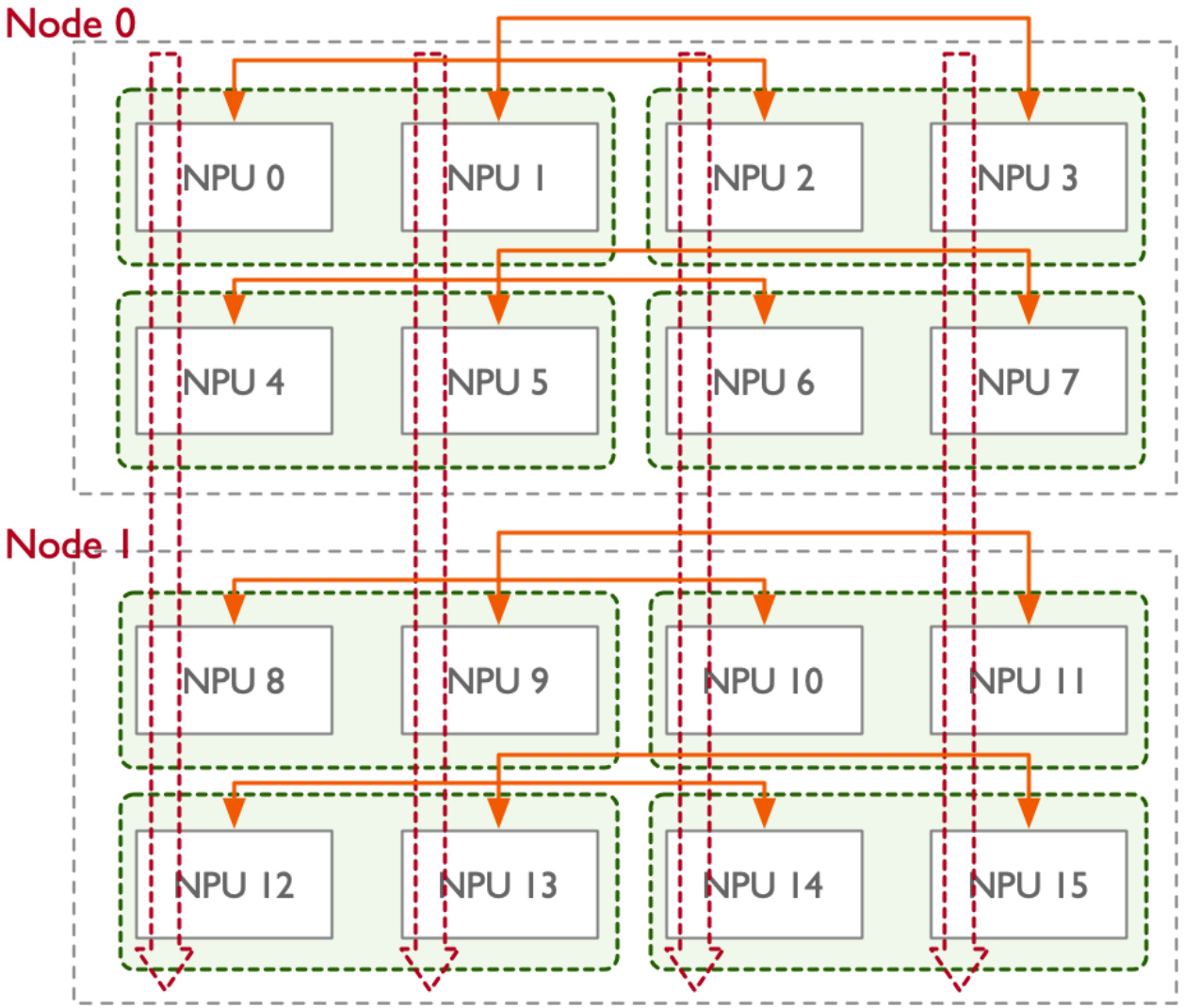
4.遍历 TP 分组 size

5.每隔 t 个取一个作为DP group中的一份

当前 rank 对应的 DP 进程组

当前 DP 进程组的 ranks se.g.,rank2 = [g1, g3]

DP 进程组 rank 分配



- world size = 16 , PP = 4 , TP = 2
- DP group16 / (4*2) = 8
  - 流水线分成 P 个 stage，每个 stage 有 n // p 个 NPU，stage i 的 rank 范围：[i * n//p, (i+1) * n//p]。e.g. rank 2 rank [0, 1,2,3]。
  - 每个 stage 中 ranks，每隔 t 取一个作为数据并行 group 中一份 → 每 DP group大小为 n // p // t = d
  - 通常在 all_reduce() 时候用到

DP 进程组使用

```python
def get_data_parallel_group(with_context_parallel=False):
    """Get the data parallel group the caller rank belongs to."""
    if with_context_parallel:
        assert (
            _DATA_PARALLEL_GROUP_WITH_CP is not None
        ), 'data parallel group with context parallel combined is not initialized'
        return _DATA_PARALLEL_GROUP_WITH_CP
```

```python
def average_losses_across_data_parallel_group(losses):
    """Reduce a tensor of losses across all GPUs."""
    averaged_losses = torch.cat(
        [loss.clone().detach().view(1) for loss in losses])
    torch.distributed.all_reduce(averaged_losses,
                                 group=mpu.get_data_parallel_group())
    averaged_losses = averaged_losses / \
        torch.distributed.get_world_size(group=mpu.get_data_parallel_group())
    return averaged_losses
```

DP 进程组使用

```
409     if wrap_with_ddp:
410         config = get_model_config(model[0])
411         model = [DDP(config,
412                      model_chunk,
413                      data_parallel_group=mpu.get_data_parallel_group(with_context_parallel=True),
414                      expert_data_parallel_group=mpu.get_data_modulo_expert_parallel_group(),
415                      accumulate_allreduce_grads_in_fp32=args.accumulate_allreduce_grads_in_fp32,
416                      overlap_grad_reduce=args.overlap_grad_reduce,
417                      use_distributed_optimizer=args.use_distributed_optimizer,
418                      # Turn off bucketing for model_chunk 2 onwards, since communication for these
419                      # model chunks is overlapped with compute anyway.
420                      disable_bucketing=(model_chunk_idx > 0),
421                      check_for_nan_in_grad=args.check_for_nan_in_loss_and_grad)
422                  for (model_chunk_idx, model_chunk) in enumerate(model)]
```

```
238     if torch.distributed.is_initialized() and \
239             mpu.get_data_parallel_world_size() > 1 and \
240             args.data_parallel_random_init:
241         rng_state_list = \
242             [None for i in range(mpu.get_data_parallel_world_size())]
243         torch.distributed.all_gather_object(
244             rng_state_list,
245             rng_state,
246             group=mpu.get_data_parallel_group())
247     else:
```

5 Megatron-LM

多维并行与配置关系

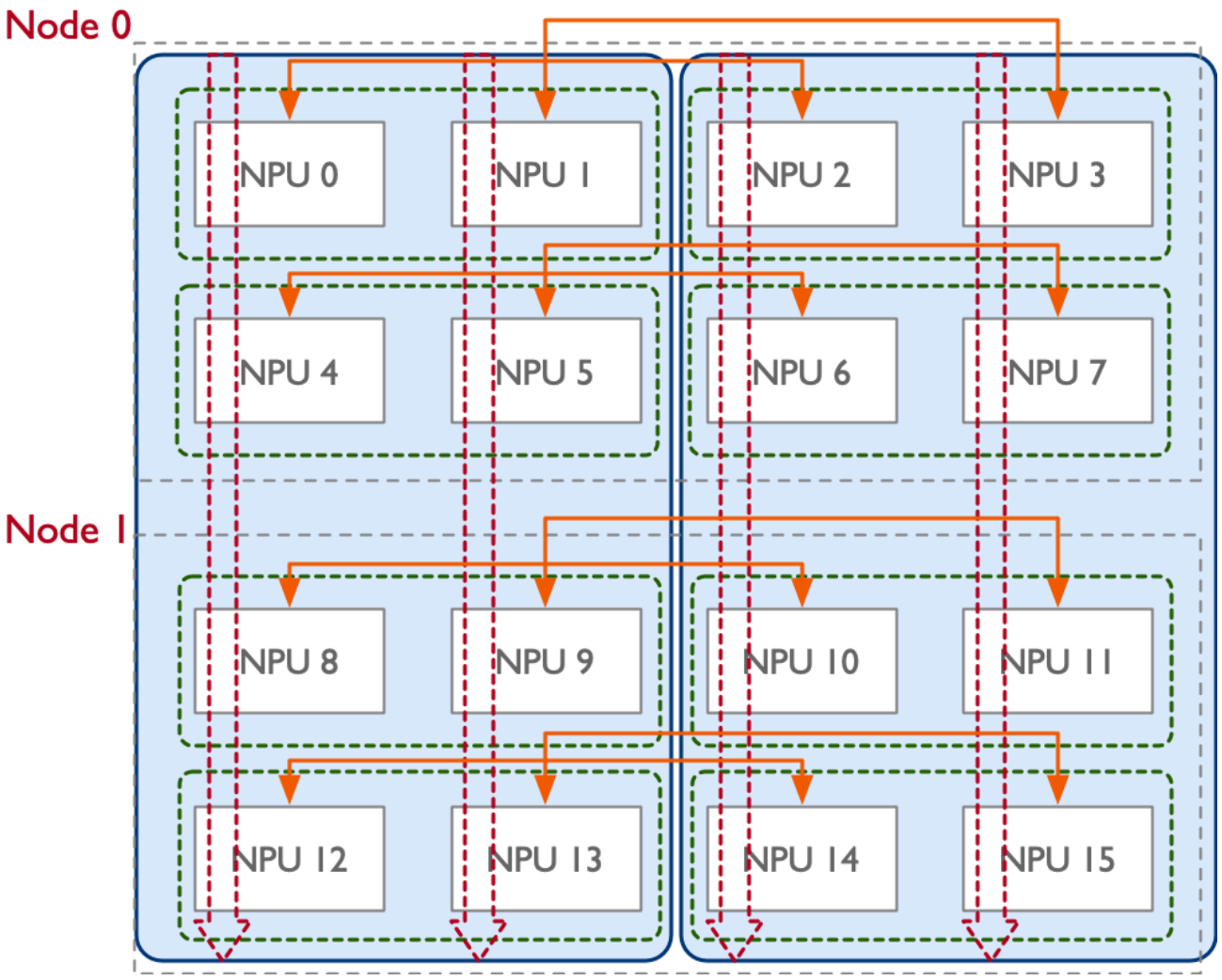
TPxPP 进程组 rank 分配

```
336
337     # Build the model-parallel groups.
338     global _MODEL_PARALLEL_GROUP
339     assert _MODEL_PARALLEL_GROUP is None, 'model parallel group is already initialized'
340     for i in range(data_parallel_size * context_parallel_size):
341         ranks = [
342             data_parallel_group_ranks_with_cp[i]
343             for data_parallel_group_ranks_with_cp in all_data_parallel_group_ranks_with_cp
344         ]
345         group = torch.distributed.new_group(
346             ranks, pg_options=get_nccl_options('mp', nccl_comm_cfgs)
347         )
348         if rank in ranks:
349             _MODEL_PARALLEL_GROUP = group
350
```

得到当前 rank 对应的模型 group

e.g., group I = [2, 3, 6, 7, 10, 11, 14, 15]

TPxPP 进程组 rank 分配



- world size = 16 ， PP = 4 ， TP = 2
- DP group16 / (4*2) = 8
  - 从 DP 域中获取第 i 个 rank 作为 MP 的 Group 中的 rank
  - 一共有 DP 个模型进程组，即对应 TP xPP 为一个大模型所需资源

PTD 并行 demo

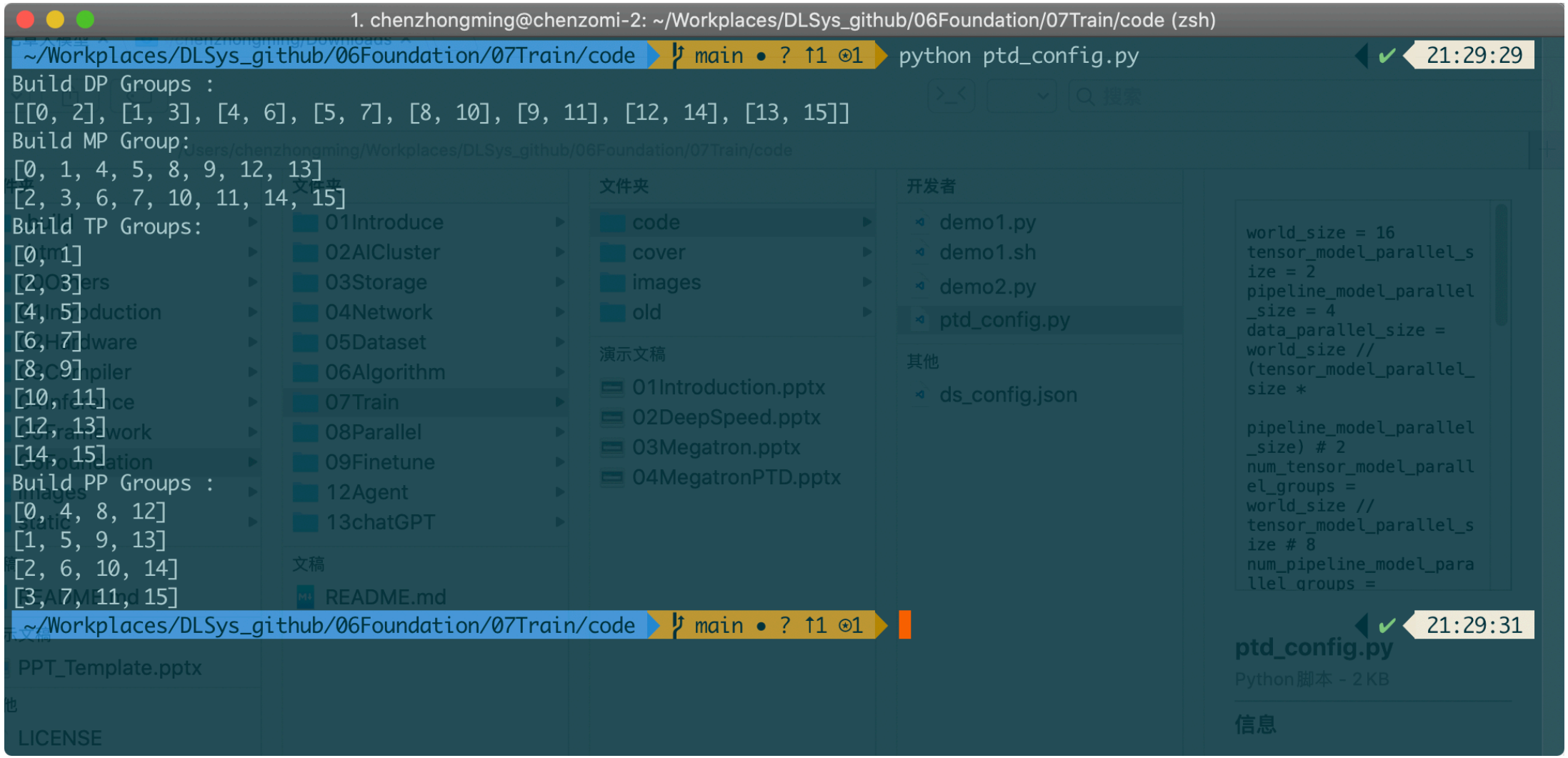
```
world_size = 16
tensor_model_parallel_size = 2
pipeline_model_parallel_size = 4
data_parallel_size = world_size // (tensor_model_parallel_size *
                                    pipeline_model_parallel_size) # 2
num_tensor_model_parallel_groups = world_size // tensor_model_parallel_size # 8
num_pipeline_model_parallel_groups = world_size // pipeline_model_parallel_size # 4
num_data_parallel_groups = world_size // data_parallel_size # 8

# Build the data-parallel groups.
print("Build DP Groups :")
all_data_parallel_group_ranks = []
for i in range(pipeline_model_parallel_size):
    start_rank = i * num_pipeline_model_parallel_groups
    end_rank = (i + 1) * num_pipeline_model_parallel_groups
    for j in range(tensor_model_parallel_size):
        ranks = range(start_rank + j, end_rank,
                      tensor_model_parallel_size)
        all_data_parallel_group_ranks.append(list(ranks))
print(all_data_parallel_group_ranks)
```

PTD 并行 demo

```
22
23  # Build the model-parallel groups.
24  print( __iter__ Group:")
25  for i in range(data_parallel_size):
26      ranks = [data_parallel_group_ranks[i]
27               for data_parallel_group_ranks in all_data_parallel_group_ranks]
28      print(list(ranks))
29
30  # Build the tensor model-parallel groups.
31  print("Build TP Groups:")
32  for i in range(num_tensor_model_parallel_groups):
33      ranks = range(i * tensor_model_parallel_size,
34                    (i + 1) * tensor_model_parallel_size)
35      print(list(ranks))
36
37  # Build the pipeline model-parallel groups and embedding groups
38  print("Build PP Groups :")
39  for i in range(num_pipeline_model_parallel_groups):
40      ranks = range(i, world_size,
41                    num_pipeline_model_parallel_groups)
42      print(list(ranks))
```

PTD 并行 demo 结果

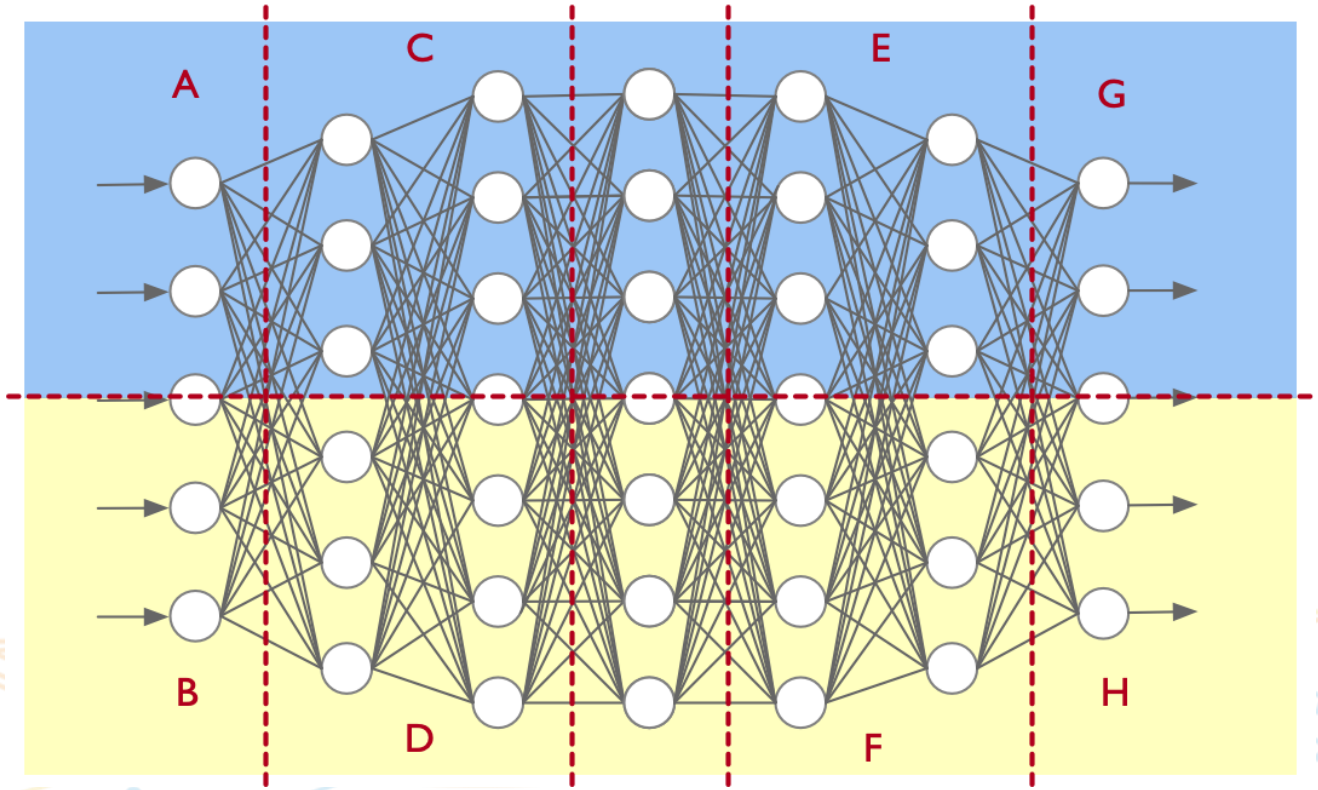


6 Megatron-LM

大模型与卡间关系

思考

1. 如何把大模型按照模型层数或者切分好的模块，分块放到对应的 NPU 上？



执行流程

1. rank 根据 PTD 全局变量映射到 NPU
2. 模型初始化通过 offset 根据 rank 生成对应层
3. 模型参数根据 rank 拷贝到对应 NPU 上然后执行训练

1. rank 根据 PTD 全局变量映射到 NPU

```
12  # Intra-layer model parallel group that the current rank belongs to.
13  _TENSOR_MODEL_PARALLEL_GROUP = N
14  # Inter-layer model parallel gr
15  _PIPELINE_MODEL_PARALLEL_GROUP =
16  # Model parallel group (both int
17  _MODEL_PARALLEL_GROUP = None
18  # Embedding group.
19  _EMBEDDING_GROUP = None
20  # Position embedding group.
21  _POSITION_EMBEDDING_GROUP = None
22  # Data parallel group that the current rank belongs to.
23  _DATA_PARALLEL_GROUP = None
24  _DATA_PARALLEL_GROUP_GLOO = None
25  # tensor model parallel group and data parallel group combined
26  # used for fp8 and moe training
27  _TENSOR_AND_DATA_PARALLEL_GROUP = None
28  # Expert parallel group that the current rank belongs to.
29  _TENSOR_AND_EXPERT_PARALLEL_GROUP = None
30  _DATA_MODULO_EXPERT_PARALLEL_GROUP = None
31  _DATA_MODULO_EXPERT_PARALLEL_GROUP_GLOO = None
```

- mpu.initialize_model_parallel() 设置 PTD 等进程组，每个 rank 对应进程都有其全局变量，进程自动映射到 NPU 上：
  - e.g.，rank 2 对应进程启动后，从初始化全局变量中获取 data_parallel group: [g0, g2]，tensor model-parallel group: [g2, g3]，pipeline model-parallel group: [g2, g6, g10, g14]。

2.模型初始化通过 offset 根据 rank 生成对应层

```
1378  class ParallelTransformer(MegatronModule):
1381      def __init__(self, config,
1545              # With 8 layers, 2 stages, and 2 virtual stages, we want an assignment of
1546              # layers to stages like (each list is a model chunk):
1547              # Stage 0: [0, 1]  [4, 5]
1548              # Stage 1: [2, 3]  [6, 7]
1549              offset = mpu.get_virtual_pipeline_model_parallel_rank() * (
1550                  config.num_layers // config.virtual_pipeline_model_parallel_size) + \
1551                  (mpu.get_pipeline_model_parallel_rank() * self.num_layers)
1552          else:
1553              # Each stage gets a contiguous set of layers.
1554              if args.model_type == ModelType.encoder_and_decoder and \
1555                      mpu.get_pipeline_model_parallel_world_size() > 1
1556                  pipeline_rank = mpu.get_pipeline_model_parallel_rank
1557                  if layer_type == LayerType.encoder:
1558                      offset = pipeline_rank * self.num_layers
1559                  else:
1560                      num_ranks_in_enc = args.pipeline_model_parallel_split_rank
1561                      offset = (pipeline_rank - num_ranks_in_enc) * self.num_layers
1562              else:
1563                  offset = mpu.get_pipeline_model_parallel_rank() * self.num_layers
```

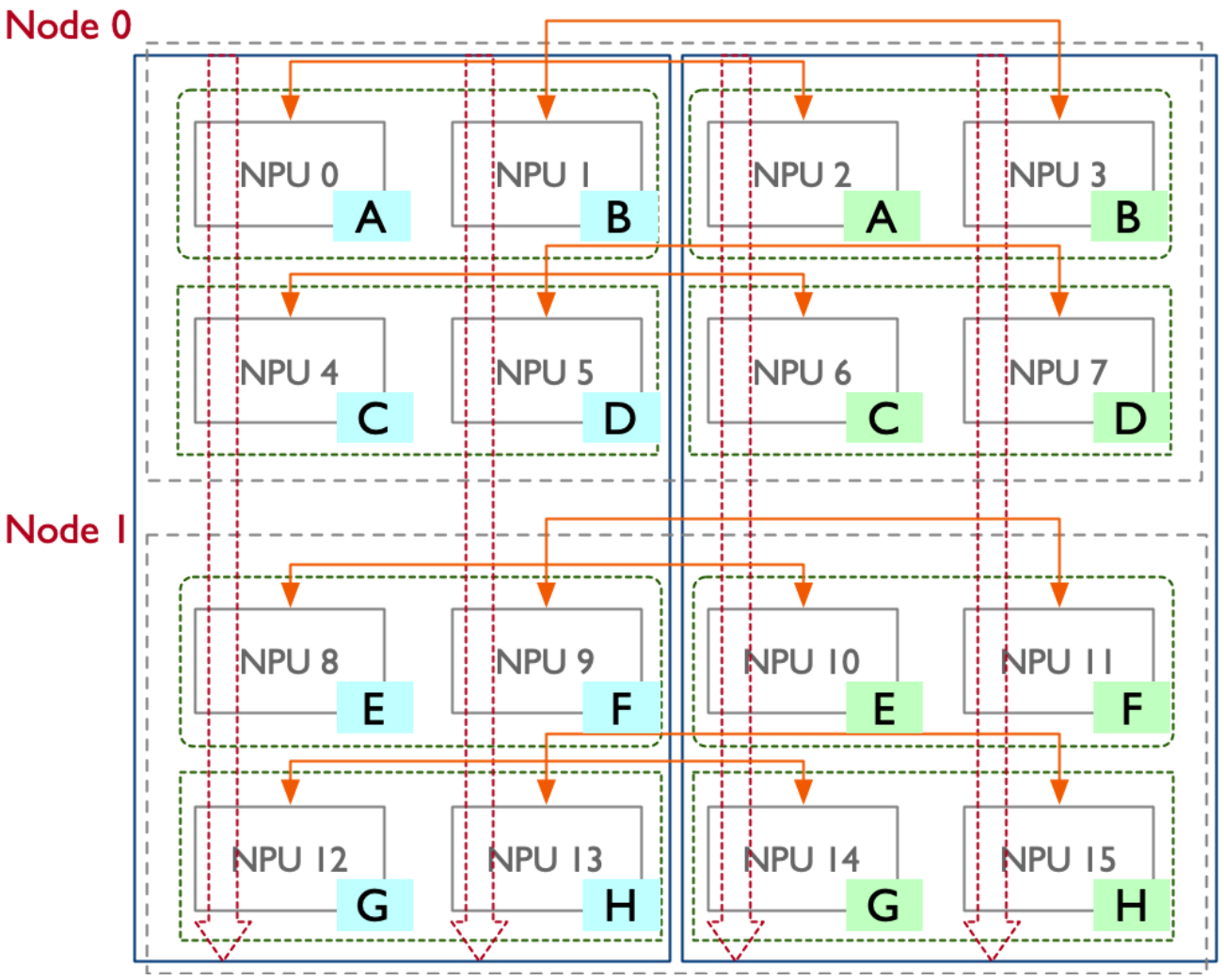
offset 根据 rank 知道自己应该生成模型所属层

2.模型初始化通过 offset 根据 rank 生成对应层

```
1571            # optimizations (e.g., pipeline output deallocation). To remedy
1572            # this, we assign a 'no-op' layer on these ranks, which will
1573            # disconnect the input tensor from the output tenso
1574            self.num_layers = 1
1575            self.layers = torch.nn.ModuleList([ NoopTransformerLayer(1) ])
1576        else:
1577            self.layers = torch.nn.ModuleList(
1578                [build_layer(i + 1 + offset) for i in range(self.num_layers)])
```

通过 self.layers 来生成对应的层

建立模型与卡间关键



1. 同名子模块具有同样的网络模型结构与参数，可进行数据并行：
   - e.g.，两个 A 可以数据并行
1. 纵向层间可进行流水线串行：
   - e.g., A → C → E → G
1. 横向是流水线的一个stage：
   - e.g., 从 0 开始，相邻为 A & B 为 TP

总结与思考

PTD 并行配置

1. TP 张量并行被用于 intra-node transformer 层：
   - TP 张量并行计算密集且耗费大量带宽，节点内利用高带宽可以高效运行。
2. PP 流水并行主要被用于inter-node transformer 层：
   - PP 通信带宽占用少，其可以有效利用集群中多卡设计
3. DP 在 PP 和 TP 基础之上进行加持，使得训练可以扩展到更大规模和更快的速度：
   - 尽管 DP 可高效扩展，但不能单独使用 DP 来训练超大模型， a）HBM 不足，b）数据并行扩展限制

PTD 并行配置

1. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism
2. Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM
3. Reducing Activation Recomputation in Large Transformer Models

Number of parameters (billion)	Attention heads	Hidden size	Number of layers	Tensor model-parallel size	Pipeline model-parallel size	Number of GPUs	Batch size	Achieved teraFlOP/s per GPU	Percentage of theoretical peak FLOP/s	Achieved aggregate petaFLOP/s
1.7	24	2304	24	1	1	32	512	137	44%	4.4
3.6	32	3072	30	2	1	64	512	138	44%	8.8
7.5	32	4096	36	4	1	128	512	142	46%	18.2
18.4	48	6144	40	8	1	256	1024	135	43%	34.6
39.1	64	8192	48	8	2	512	1536	138	44%	70.8
76.1	80	10240	60	8	4	1024	1792	140	45%	143.8
145.6	96	12288	80	8	8	1536	2304	148	47%	227.1
310.1	128	16384	96	8	16	1920	2160	155	50%	297.4
529.6	128	20480	105	8	35	2520	2520	163	52%	410.2
1008.0	160	25600	128	8	64	3072	3072	163	52%	502.0

Table 1: Weak-scaling throughput for GPT models ranging from 1 billion to 1 trillion parameters.

Thank you

把AI系统带入每个开发者、每个家庭、每个组织，构建万物互联的智能世界

Bring AI System to every person, home and organization for a fully connected, intelligent world.

ZOMI

Course chenzomi12.github.io

GitHub github.com/chenzomi12/DeepLearningSystem