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

2.1 DeepSpeed

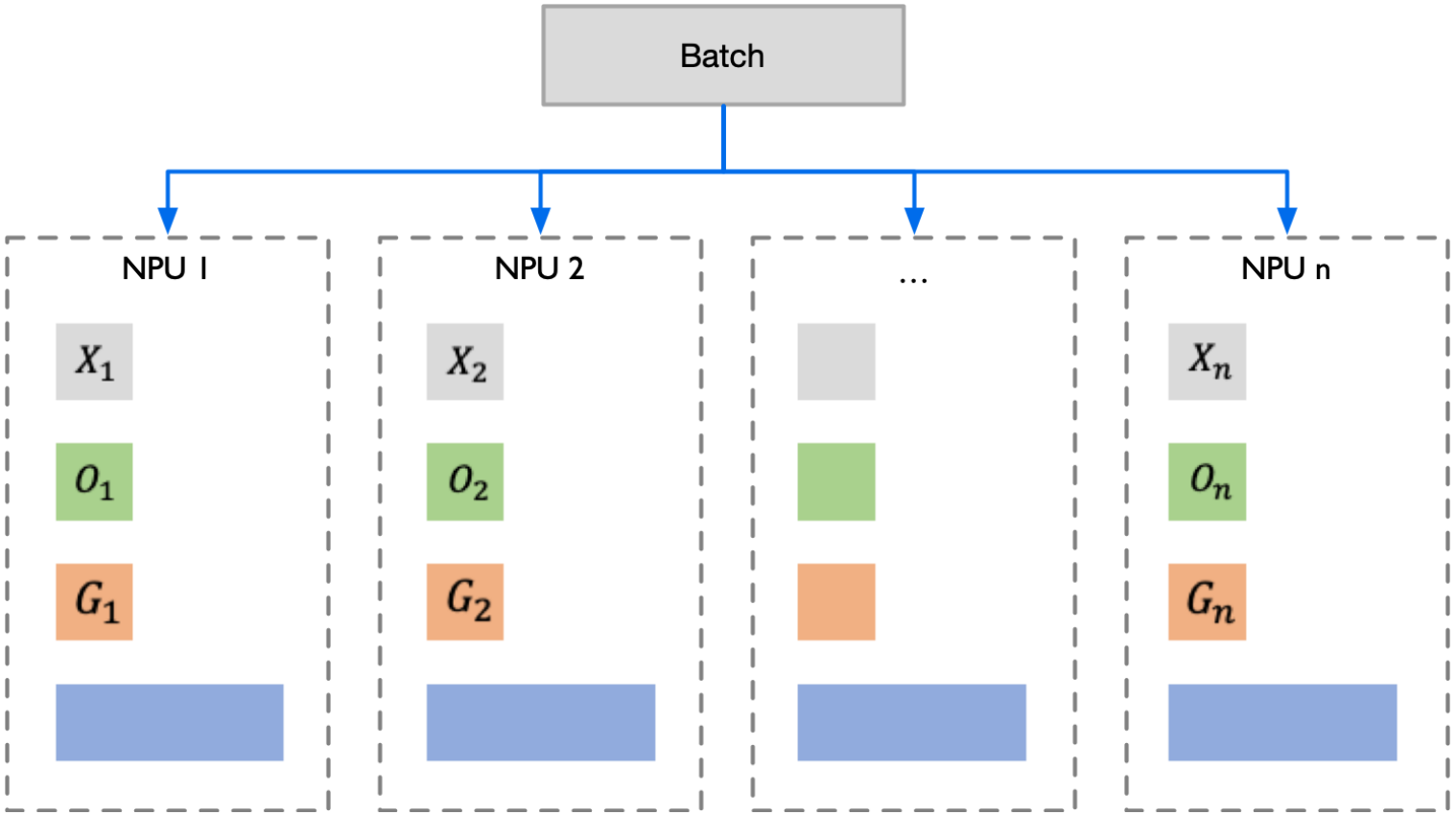
ZeRO-1 原理

ZeRO Stage 1 算法原理

1. 从 optimizer state 开始，batch 数据分成 N 份，每块 NPU 一份；
2. 执行I Step forward & backward后，各得一份梯度 G_n ；
3. 对梯度 G_n 执行All-Reduce，得到完整梯度 G
   - 单 NPU 通讯量 2Φ ；
4. 得到完整梯度 G，对权重 W 更新，而权重 W 更新由 optimizer states 和 grad 共同决定
   - 每块 NPU 有部分 W，对 W 执行 All-Gather，从其他 NPU 上更新好的部分
   - 产生单 NPU 通讯量 Φ

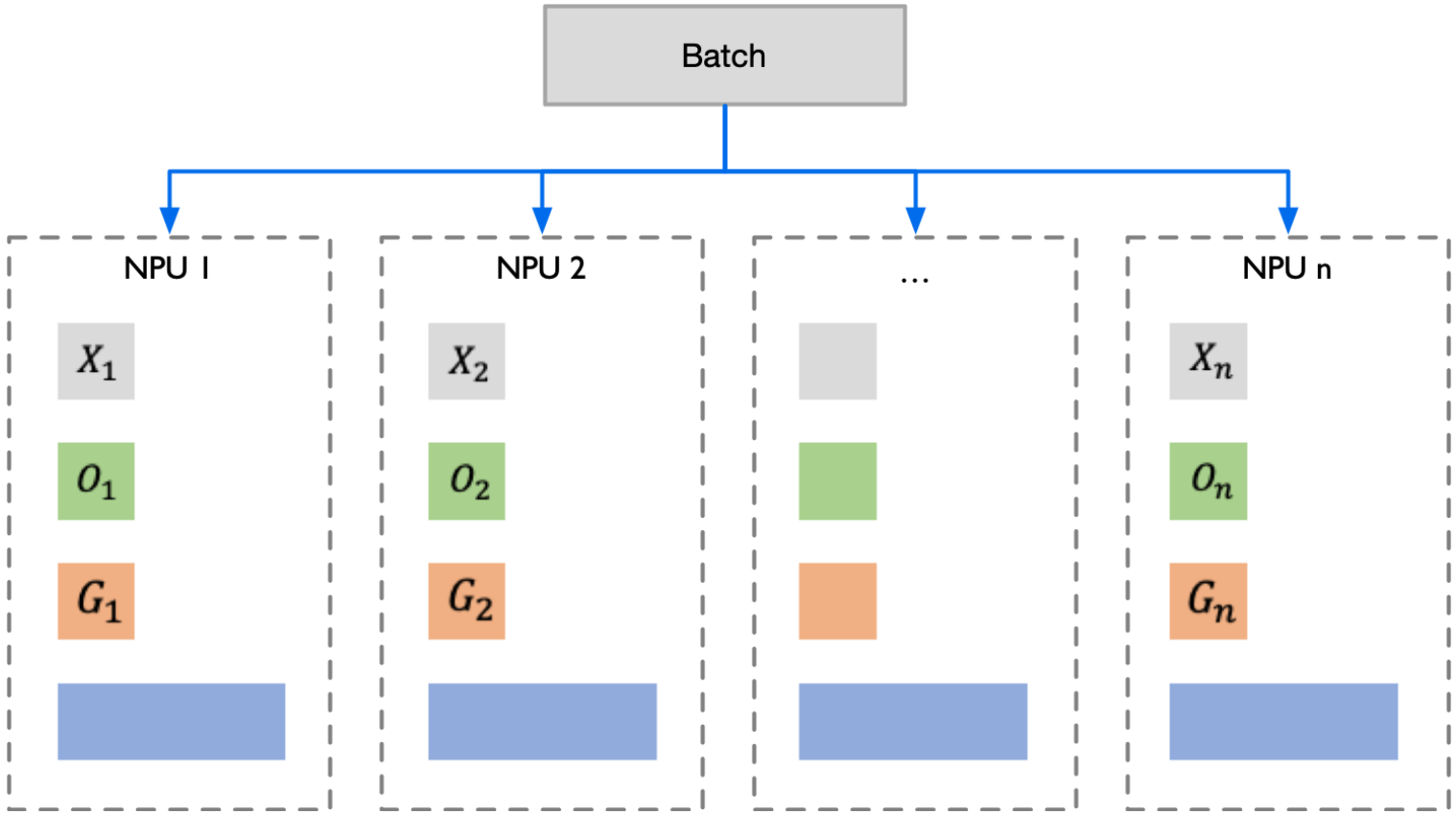
ZeRO Stage 1 原理

- 1. 从 optimizer state 开始，batch 数据分成 N 份，每块 NPU 一份；



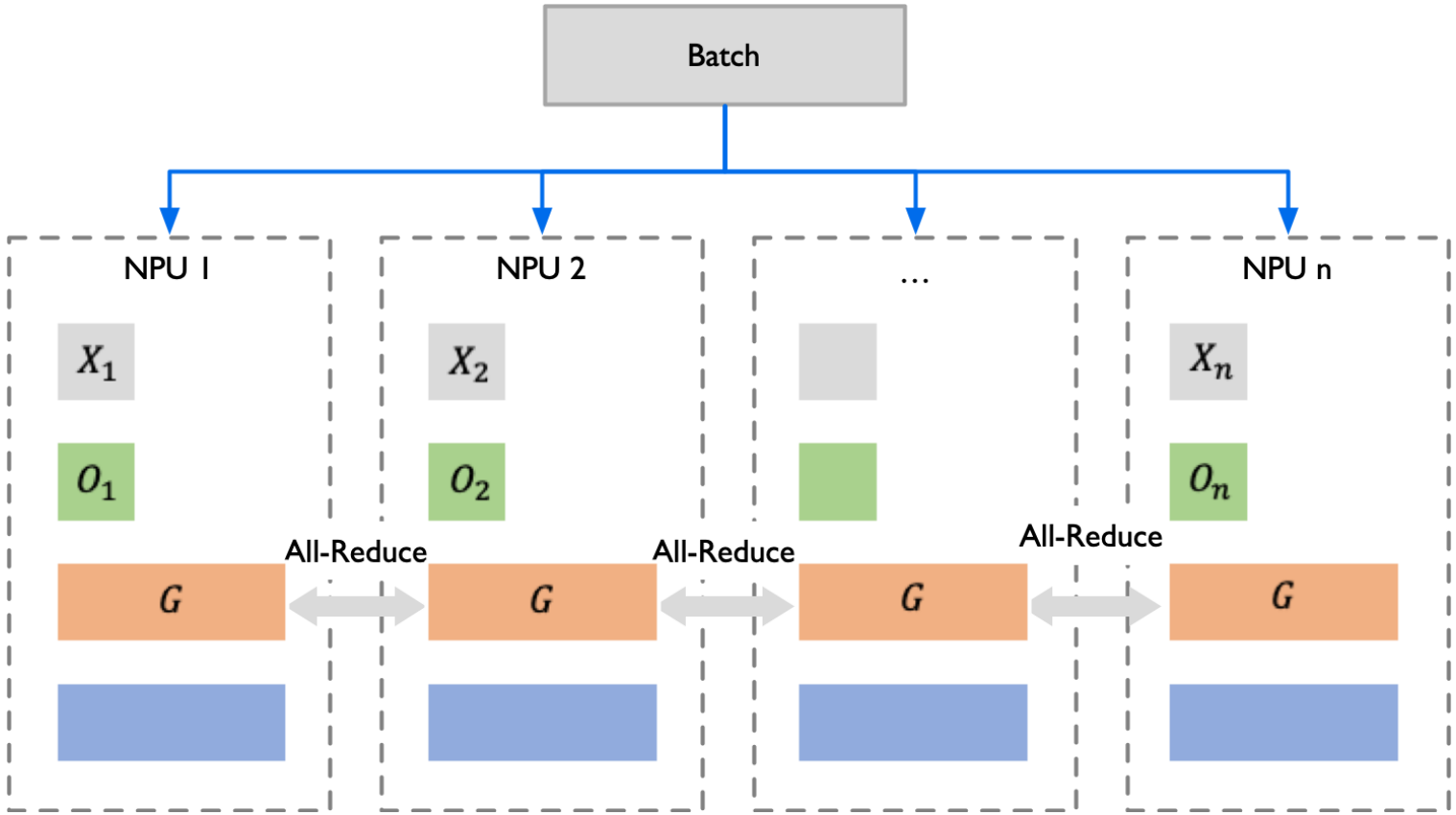
ZeRO Stage 1 原理

- 1. 从 optimizer state 开始，batch 数据分成 N 份，每块 NPU 一份；
- 2. 执行 1 步 Step forward & backward 计算后，各得一份梯度 G_n；



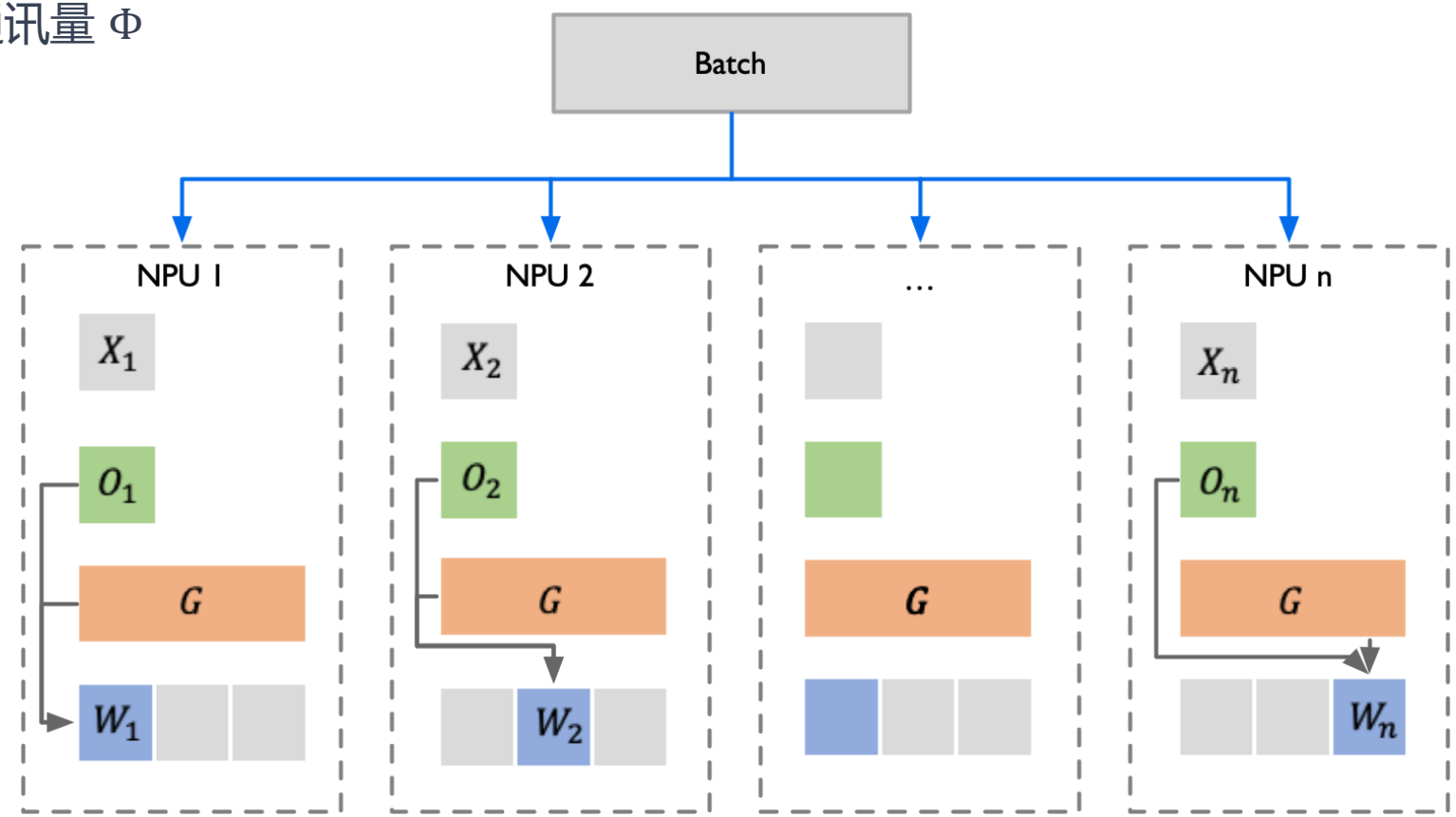
ZeRO Stage 1 原理

- 3. 对梯度 G_n 执行All-Reduce，得到完整梯度 ；
  - 单 NPU 通讯量 2Φ ；



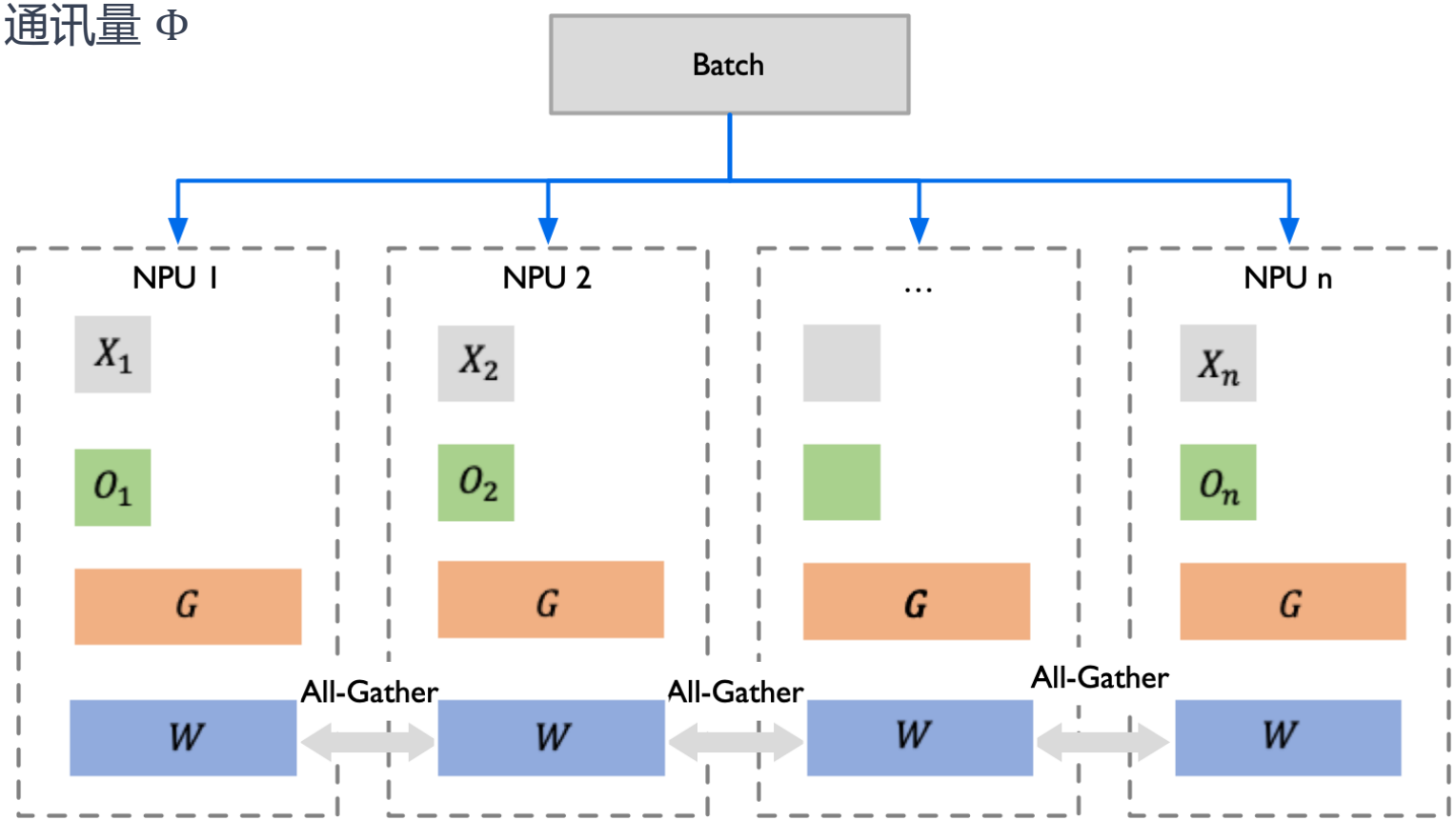
ZeRO Stage 1 原理

- 4. 得到完整梯度 G，对权重 W 更新，而权重 W 更新由 optimizer states 和 grad 共同决定
  - 每块 NPU 有部分 W，对 W 执行 All-Gather，从其他 NPU 上更新好的部分 W；
  - 产生单 NPU 通讯量 Φ



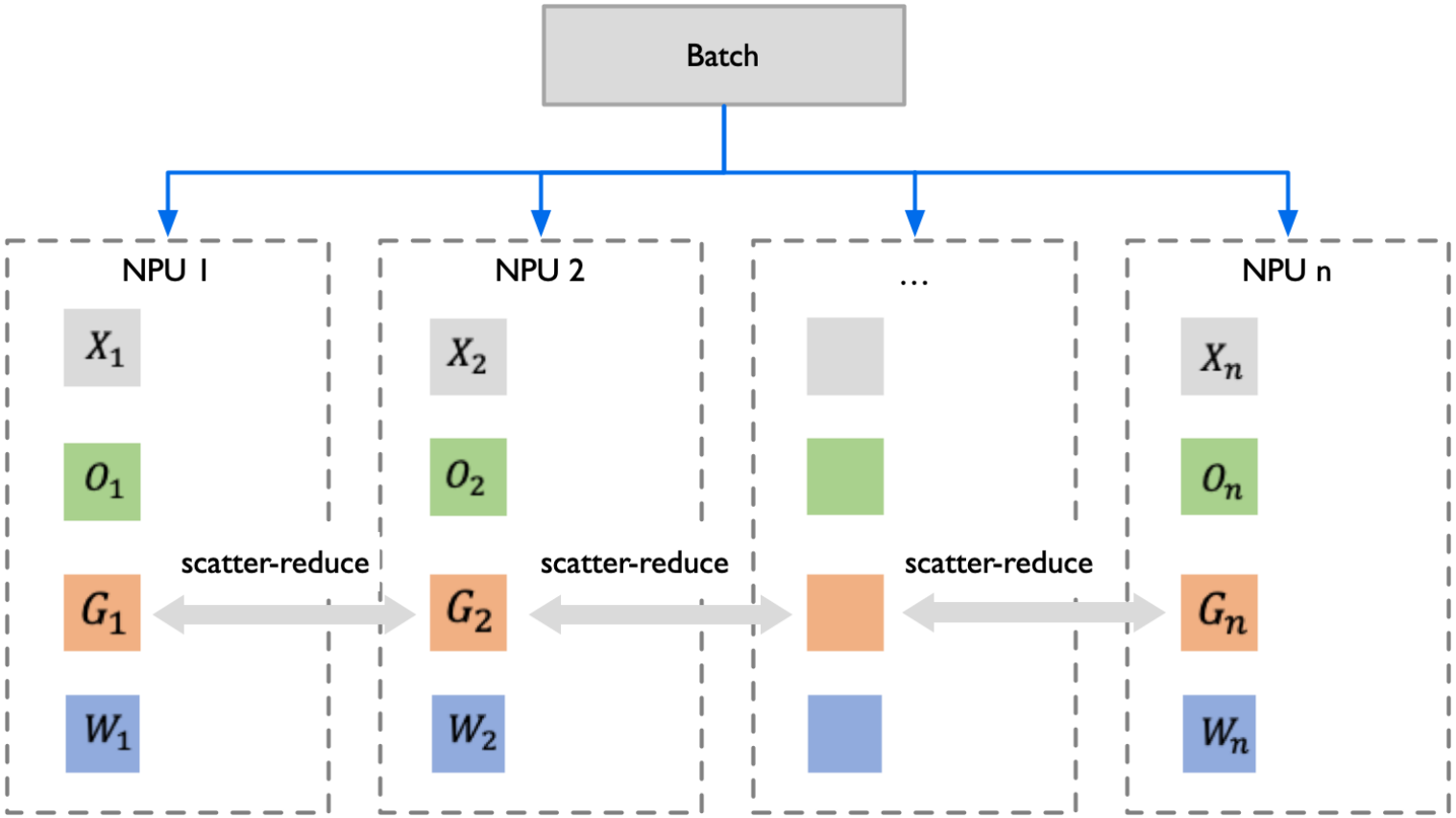
ZeRO Stage 1 原理

- 4. 得到完整梯度 G，对权重 W 更新，而权重 W 更新由 optimizer states 和 grad 共同决定
  - 每块 NPU 有部分 W，对 W 执行 All-Gather，从其他 NPU 上更新好的部分 W；
  - 产生单 NPU 通讯量 Φ



ZeRO Stage 1 实际执行

- 3. 对梯度 G_n 执行 scatter-reduce，各 NPU 维护 Optimizer 更新对应 W_n
- 4. 再对权重 W_n 执行 all-gather 使得每 NPU 都有更新后完整 W，最终通讯量为 2Φ

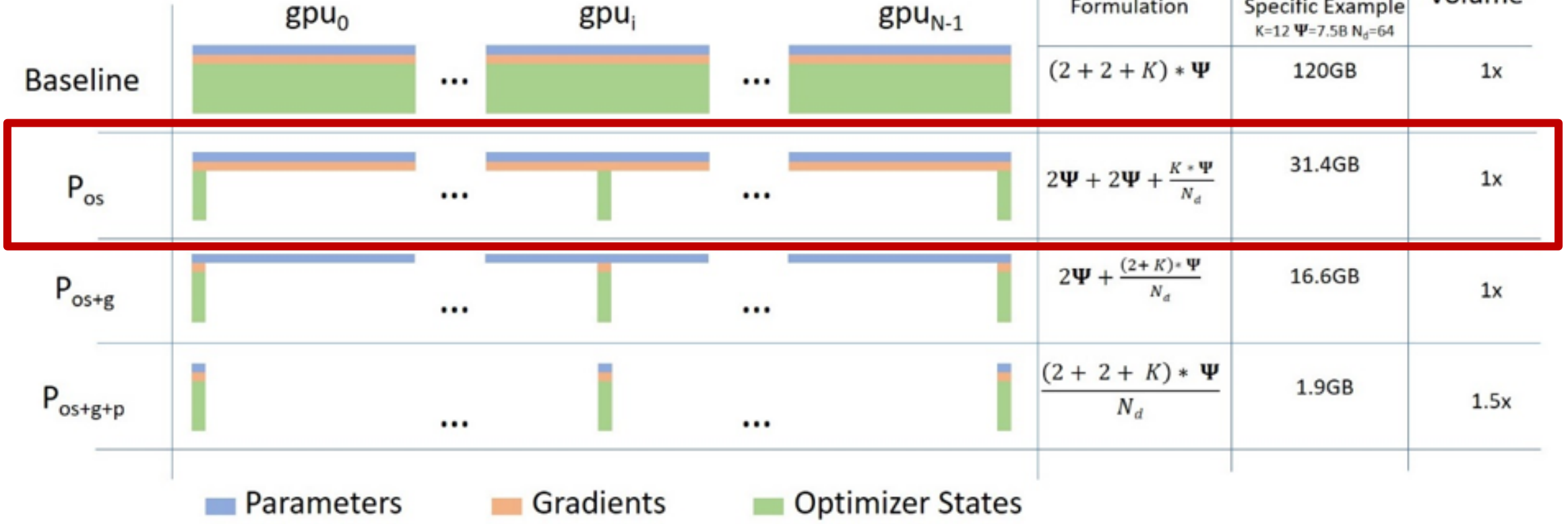


ZeRO Stage 1 计算

- 在 P_{os} 阶段优化器状态划分：
  - 根据 DP 维度 N_d 将 Adam 优化器状态划分为 N_d 等份；
  - 每 NPU 需要存储和更新总优化器状态的 $1/N_d$，并更新 $1/N_d$ 参数；
  - 每训练 Step 末尾，使用 all-gather 获得整个参数的更新完整的权重 W_n；

- 显存分析：
  - 显存从 4Ψ + KΨ 降低到 4Ψ + KΨ/ N_d
  - 当 N_d 很大时，显存占用接近于 4Ψ，带来 4 倍显存节约。

ZeRO Stage 1 计算



	Memory Consumption		Comm Volume
	Formulation	Specific Example K=12 Ψ=7.5B N_d=64	
Baseline	$(2+2+K)*\Psi$	120GB	1x
P_{os}	$2\Psi+2\Psi+\frac{K*\Psi}{N_d}$	31.4GB	1x
P_{os+g}	$2\Psi+\frac{(2+K)*\Psi}{N_d}$	16.6GB	1x
P_{os+g+p}	$\frac{(2+2+K)*\Psi}{N_d}$	1.9GB	1.5x

2.2 DeepSpeed

ZeRO-2 原理

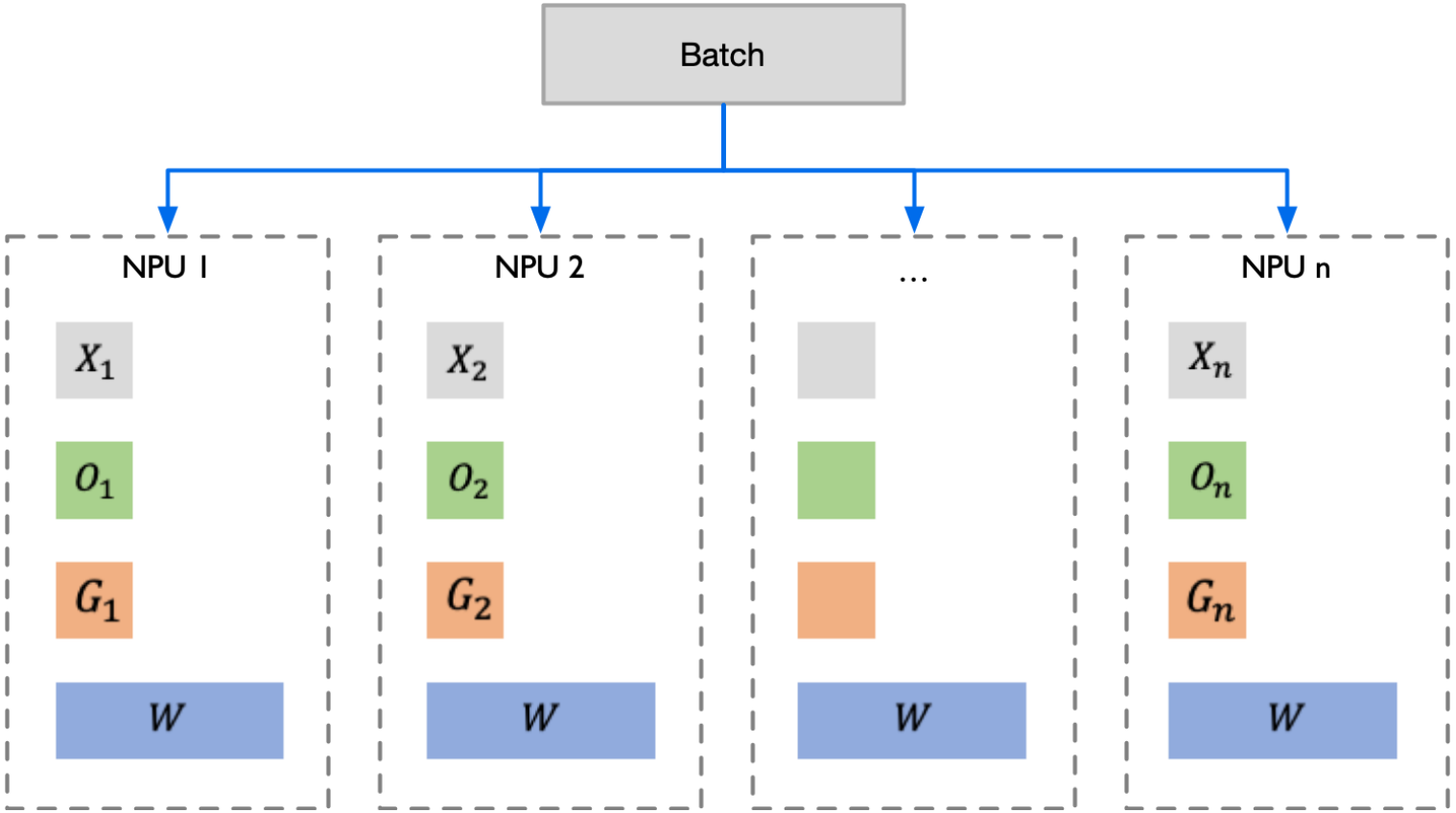
ZeRO Stage 2 原理

- 梯度 Grad 也进行切分，每个 NPU 各自维护一块梯度。

1. 从 optimizer state 开始，batch 数据分成 N 份，每块 NPU 一份
2. 执行 One Step forward & backward 后，各得一份梯度 G_n
3. 对梯度 G_n 执行 Reduce-Scatter，保证每个 NPU 所维持的梯度 G_n 是聚合梯度：
   - e.g. 对 NPU1 负责维护梯度 G_1 ，其他 NPU 只需要把 G_1 对应位置梯度发给 NPU1 即可；
   - 汇总完毕后白色块对 NPU1 无用从显存移除，单卡通讯量 Φ；
4. 每 NPU O_n 和 梯度 G_n 更新相应 W_n，即每块 NPU 维护独立的权重 W_n
5. 最后对权重 W_n 执行 All-Gather，将其他 NPU 的 W_n 同步一份完整 W，单卡通讯量 Φ

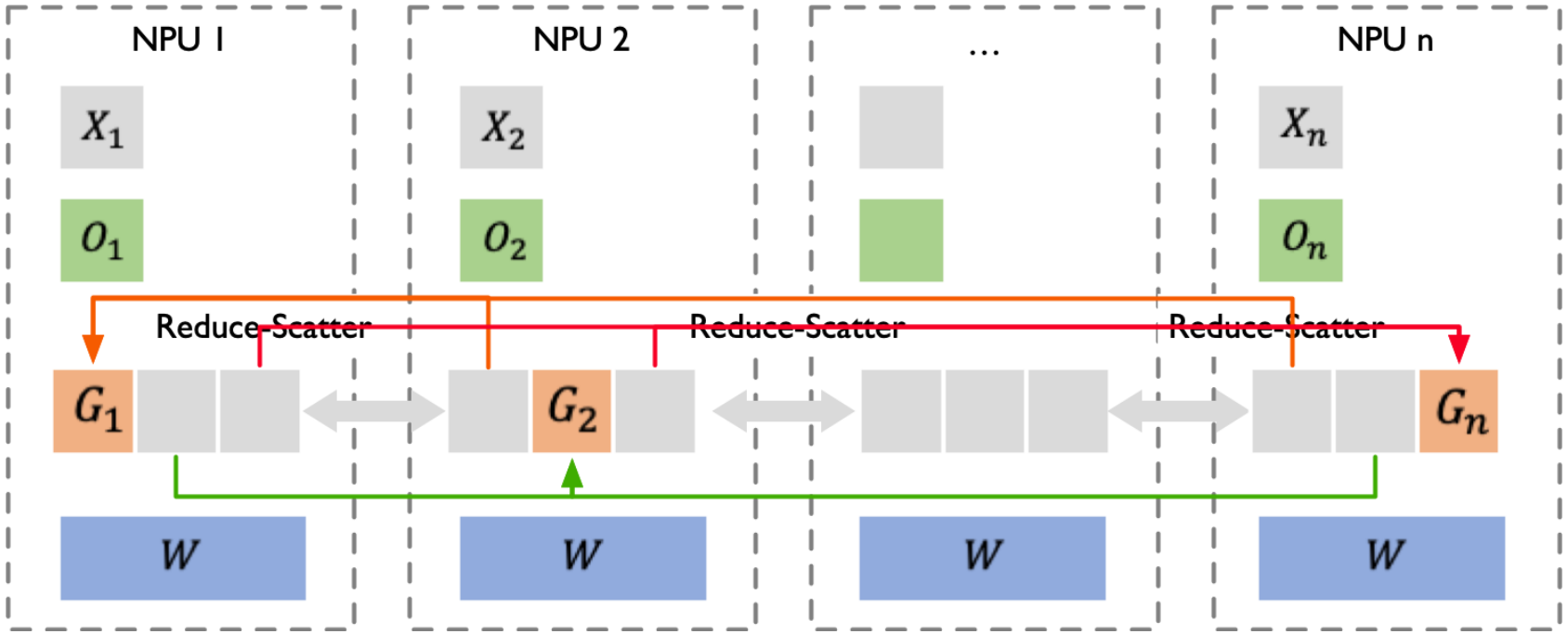
ZeRO Stage 2 原理

- 1. 从 optimizer state 开始，batch 数据分成 N 份，每块 NPU 一份
- 2. 执行 One Step forward & backward 后，各得一份梯度 G_n



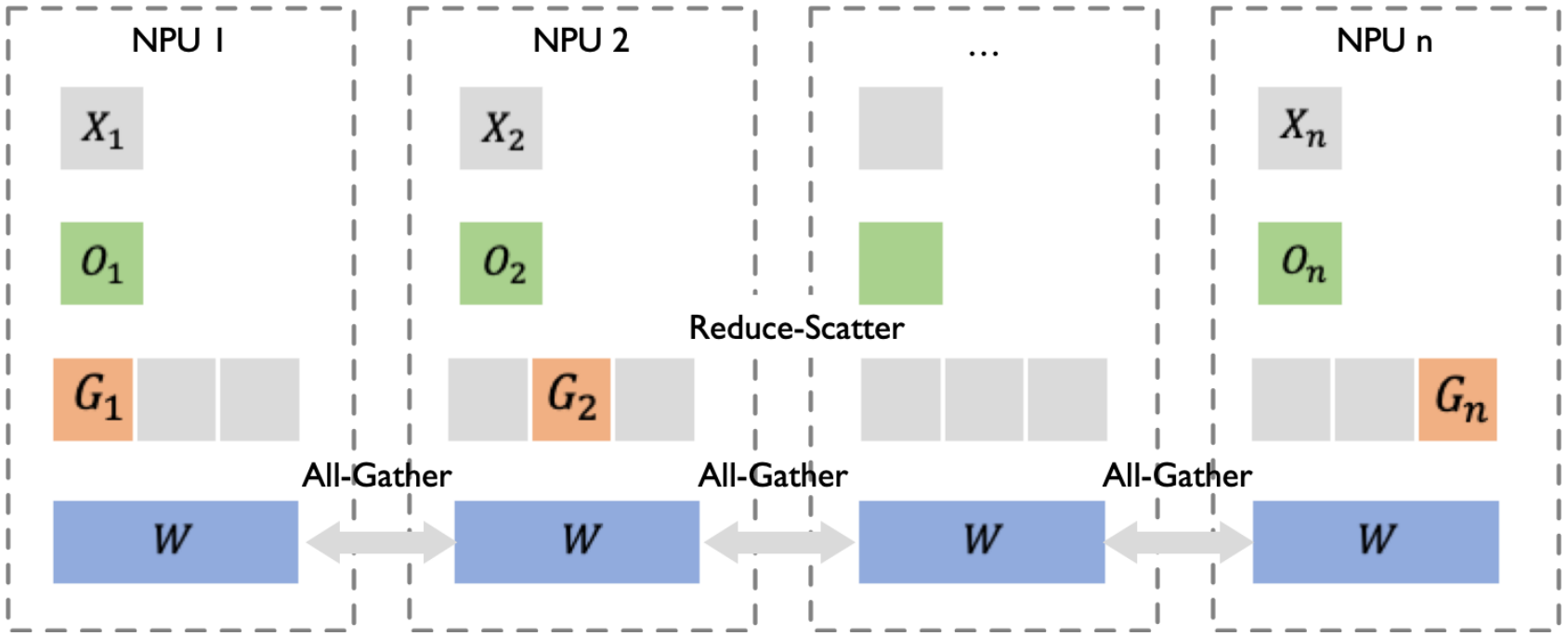
ZeRO Stage 2 原理

- 3. 对梯度 G_n 执行 Reduce-Scatter，保证每个 NPU 所维持的梯度 G_n 是聚合梯度：
  - e.g. 对 NPU1 负责维护梯度 G_1 ，其他 NPU 只需要把 G_1 对应位置梯度发给 NPU1 即可；
  - 汇总完毕后白色块对 NPU1 无用从显存移除，单卡通讯量 Φ；



ZeRO Stage 2 原理

- 4. 每 NPU O_n 和 梯度 G_n 更新相应 W_n，即每块 NPU 维护独立的权重 W_n
- 5. 最后对权重 W_n 执行 All-Gather，将其他 NPU 的 W_n 同步一份完整 W，单卡通讯量 Φ

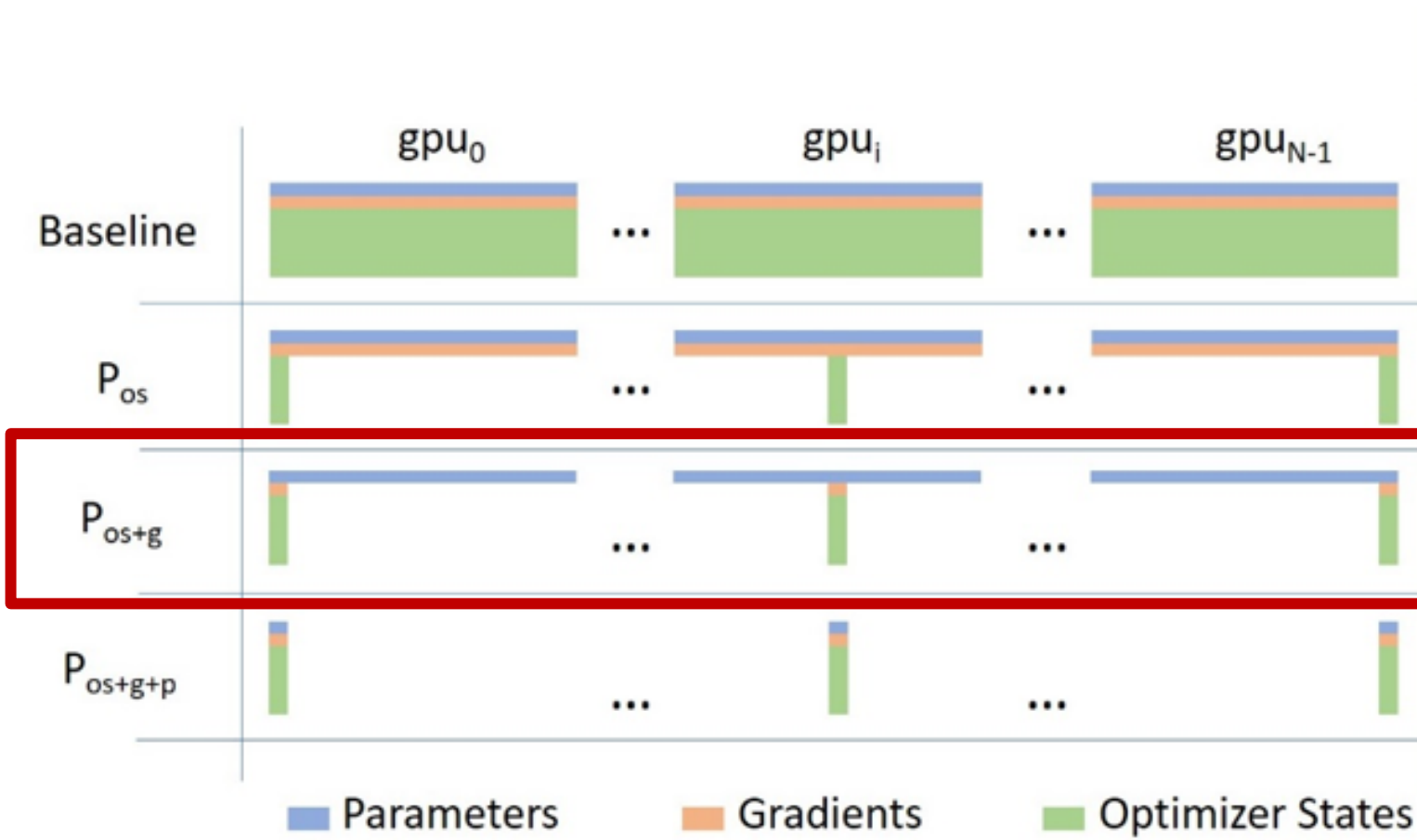


ZeRO Stage 2 计算

- 在 P_g 阶段优化器状态 + 梯度划分：
  - 梯度跟优化器强相关，因此优化器可以更新其独立的梯度参数；
  - 重点是更新梯度参数时候使用 Reduce-Scatter，梯度参数更新后马上释放，显存从 2Ψ → 2Ψ/ N_d ；
  - 具体实现：实现过程中使用分桶 Bucket，将梯度分到 Bucket 中并在桶上进行 Reduce-Scatter；

- 显存分析：
  - 移除梯度和优化器状态冗余，将显存从 4Ψ + KΨ 降低到 2Ψ + KΨ/ N_d
  - 当 N_d 很大时，显存占用接近于 2Ψ，带来 8 倍显存节约。

ZeRO Stage 2 计算



	Memory Consumption		Comm Volume
	Formulation	Specific Example K=12 Ψ=7.5B N_d=64	
Baseline	$(2+2+K)*\Psi$	120GB	1x
P_{os}	$2\Psi+2\Psi+\frac{K*\Psi}{N_d}$	31.4GB	1x
P_{os+g}	$2\Psi+\frac{(2+K)*\Psi}{N_d}$	16.6GB	1x
P_{os+g+p}	$\frac{(2+2+K)*\Psi}{N_d}$	1.9GB	1.5x

2.3 DeepSpeed

ZeRO-3 原理

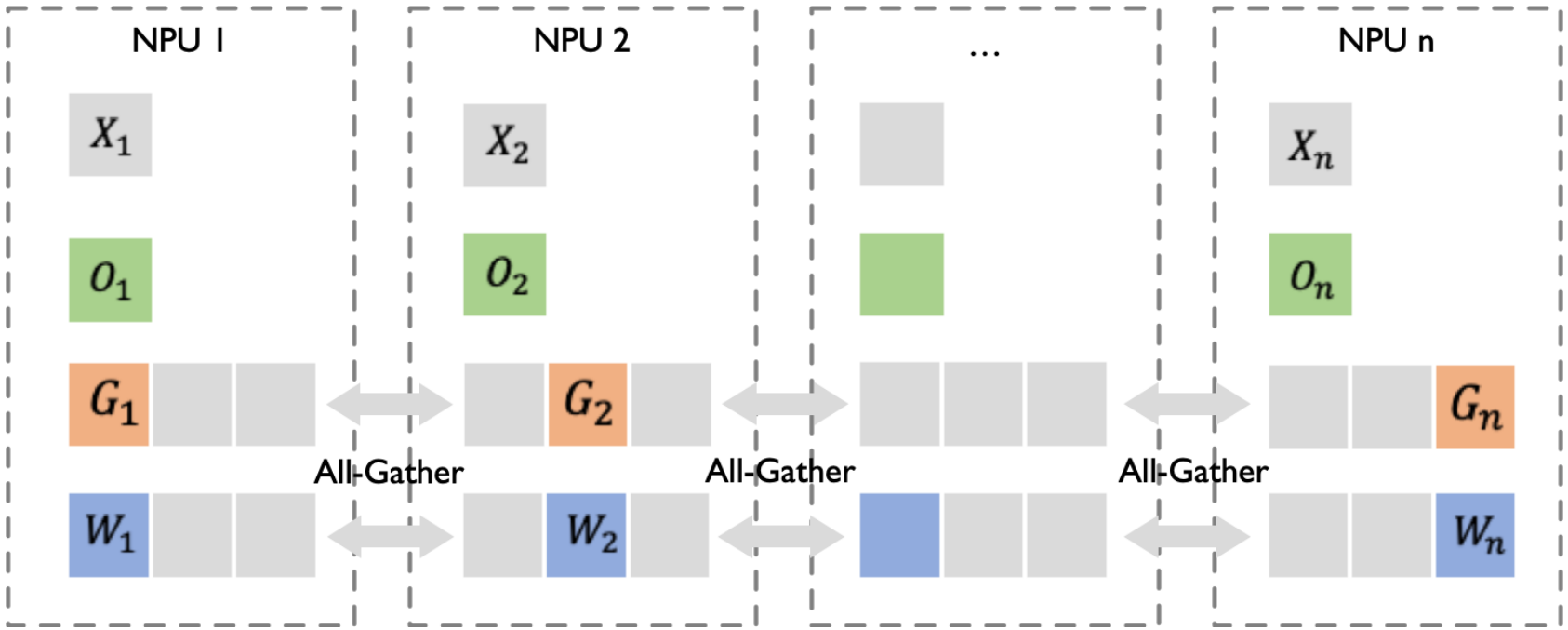
ZeRO Stage 3 原理

1. 将 batch 数据分成 N 份，每块 NPU 一份
2. Forward 计算，对 W_n 执行 All-Gather 取回分布在各 NPU上的权重 W_n 得到完整 W，并把不属于自身的权重 W_{others} 抛弃 → 单卡通讯量 Φ
3. Backward 计算，对 W_n 执行 All-Gather，取回完整 W，并把不属于自身权重 W_{others} 抛弃 → 单卡通讯量 Φ
4. backward 后得到各自梯度 G_n，对 G_n 执行 Reduce-Scatter，从其他 NPU 聚合自身维护的梯度 G_n。聚合操作结束后，立刻把不是自己维护的G抛弃 → 单卡通讯量 Φ
5. 每 NPU 只保存其权重参数 W_n，由于只维护部分参数 W_n，因此无需对 W_n 执行 All-Reduce

ZeRO Stage 3 原理

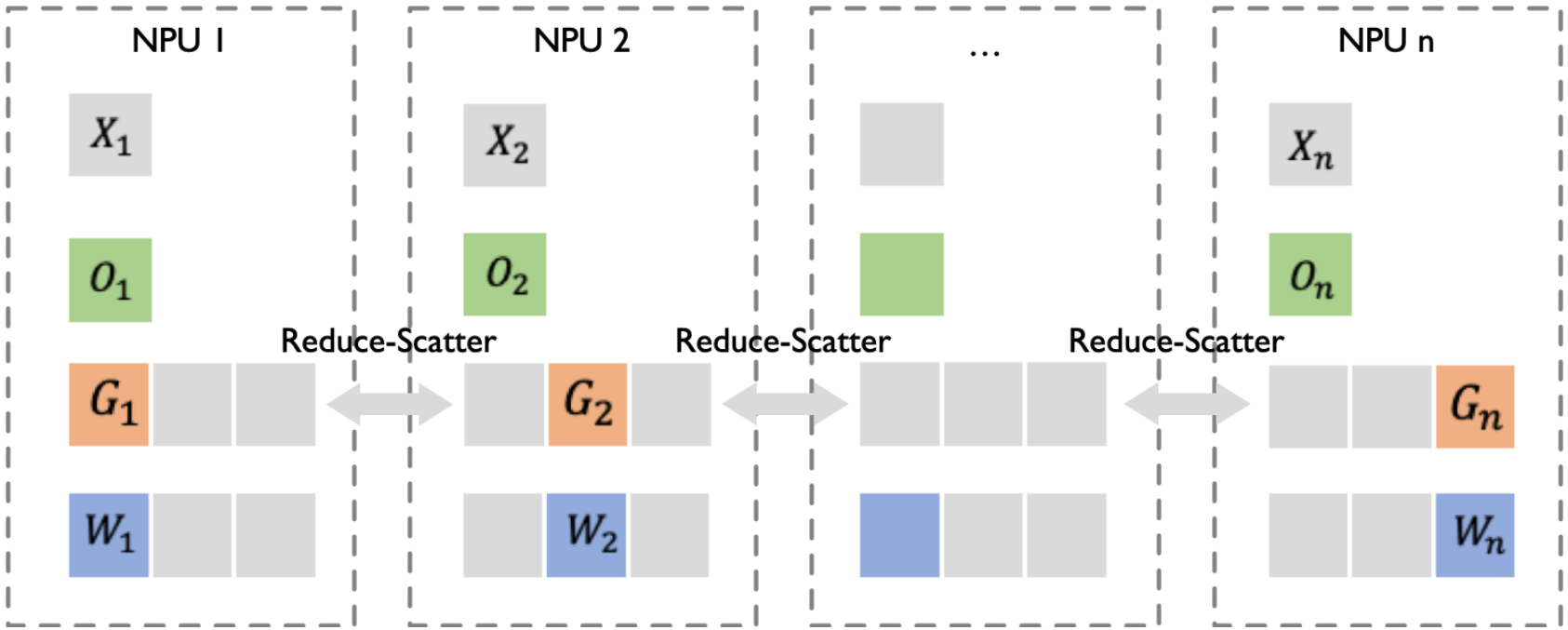
2. Forward 计算，对 W_n 执行 All-Gather 取回分布在各 NPU上的权重 W_n 得到完整 W，并把不属于自身的权重 W_{others} 抛弃 → 单卡通讯量 Φ

3. Backward 计算，对 W_n 执行 All-Gather，取回完整 W，并把不属于自身权重 W_{others} 抛弃 → 单卡通讯量 Φ



ZeRO Stage 3 原理

4. backward 后得到各自梯度 G_n，对 G_n 执行 Reduce-Scatter，从其他 NPU 聚合自身维护的梯度 G_n。聚合操作结束后，立刻把不是自己维护的G抛弃 → 单卡通讯量 Φ
5. 每 NPU 只保存其权重参数 W_n，由于只维护部分参数 W_n，因此无需对 W_n 执行 All-Reduce

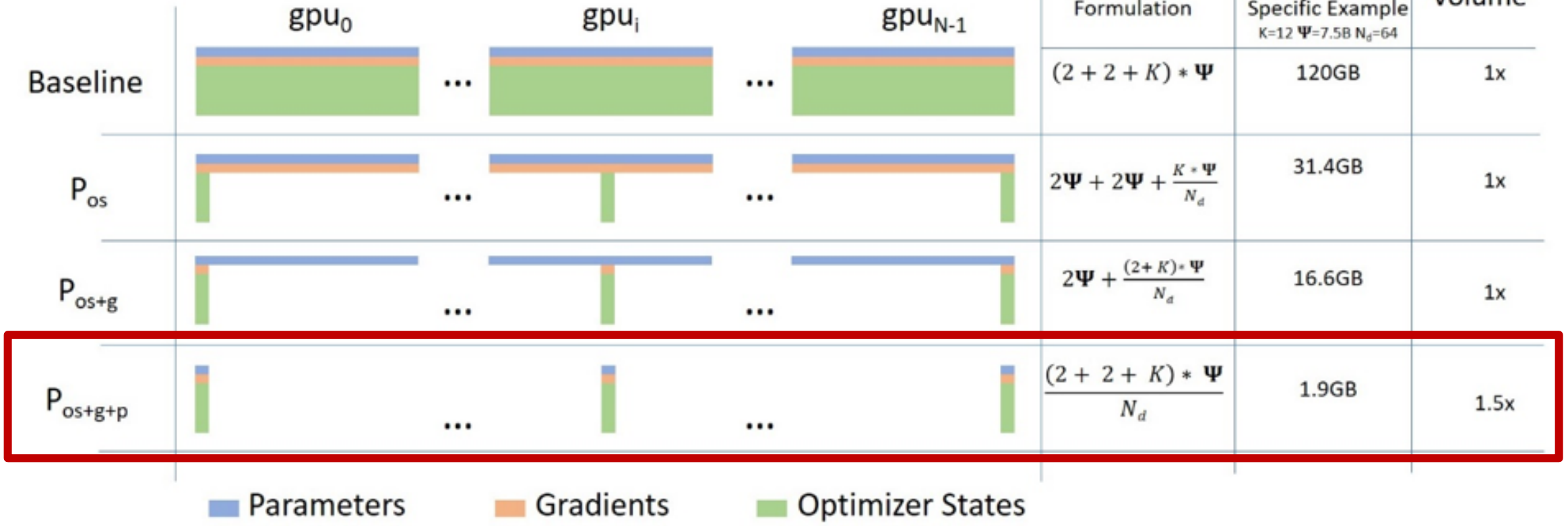


ZeRO Stage 3 计算

- 在 P_p 阶段优化器状态 + 梯度 + 权重划分：
  - 拆分到 forward & backward 过程，通过 broadcast 从其他 NPU 中获取参数；
  - 通过增加通信开销，减少每张 NPU 中的显存占用，以通信换显存，使得显存占用与 N_d 成正比；

- 显存分析：
  - 移除梯度、优化器状态、权重冗余，将显存从 4Ψ + KΨ 降低到 (4Ψ + KΨ)/ N_d
  - 带来 DP 增加 1.5X 单卡通讯量

ZeRO Stage 3 计算



	Memory Consumption		Comm Volume
	Formulation	Specific Example K=12 Ψ=7.5B N_d=64	
Baseline	$(2+2+K)*\Psi$	120GB	1x
P_{os}	$2\Psi+2\Psi+\frac{K*\Psi}{N_d}$	31.4GB	1x
P_{os+g}	$2\Psi+\frac{(2+K)*\Psi}{N_d}$	16.6GB	1x
P_{os+g+p}	$\frac{(2+2+K)*\Psi}{N_d}$	1.9GB	1.5x

ZeRO vs TP

1. ZeRO 居然把权重参数 W 给切（并行）了？那不是应该属于模型并行吗？

   ZeRO1/2/3 可以跟 TP 一起混用？

ZeRO vs TP

- ZeRO 形式上模型并行，实质上数据并行：
  1. 张量并行：相同输入 X，每 NPU 上各算模型一部分，最后通过通信来进行聚合 → forward 和 backward 过程中，NPU 只需要维护其独立 W_n 来计算即可
  2. ZeRO 并行：forward 和 backward 过程中，把 NPU 上维护 W_n 进行聚合，本质上用完整 W 进行计算。它是不同输入 X，完整参数 W，最终再进行聚合。

- 实际并行时候，只能使用 ZeRO Stage 1 + PTD 多维混合并行；

2.4 DeepSpeed

ZeRO-R 原理

ZeRO-Offload

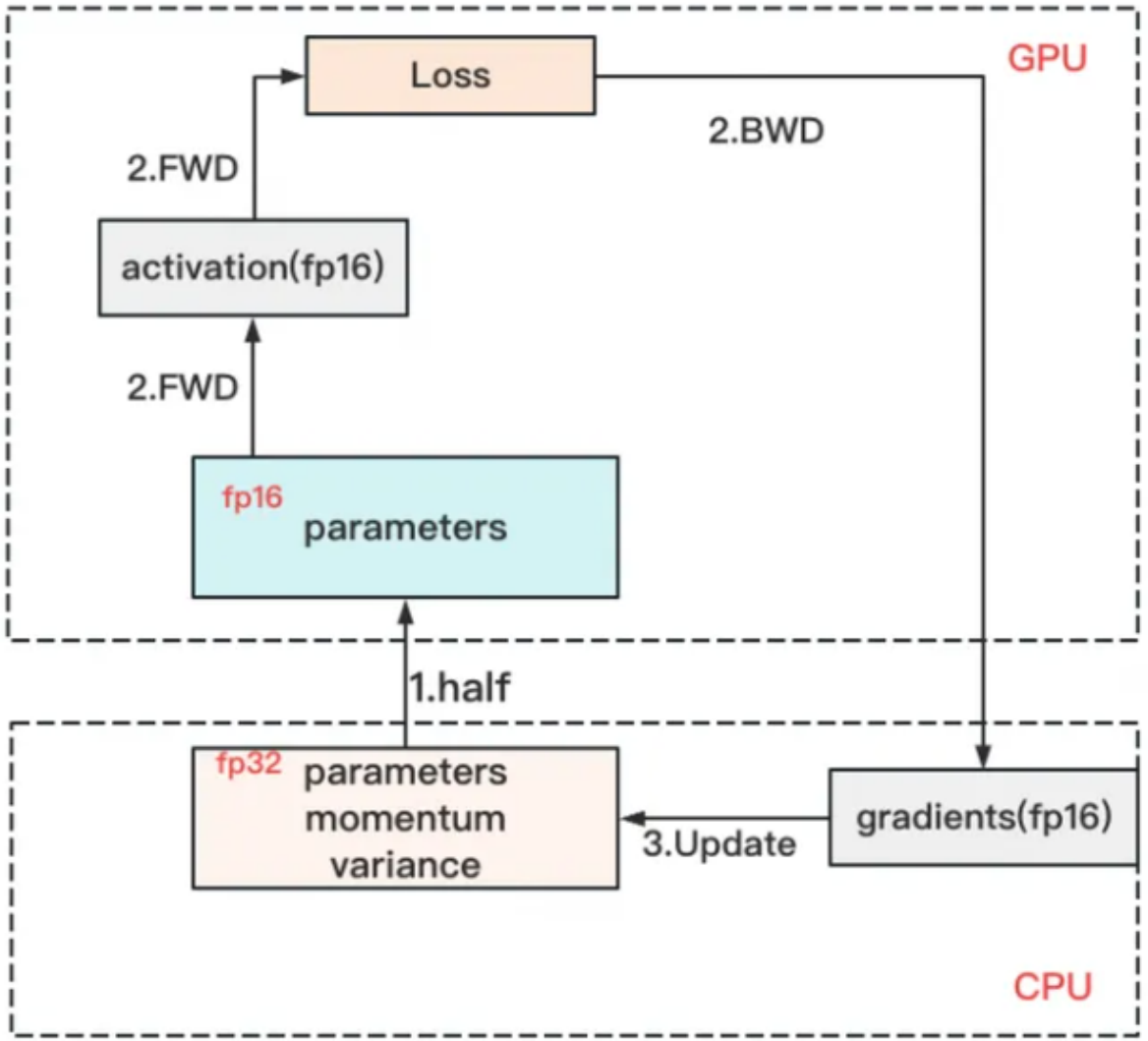
- 把占用显存多的部分卸载 offload 到CPU上，计算和激活值部分放到 NPU 上，这样比起跨机，更能节省内存，也能减少跨机跨通信域通讯压力！

ZeRO-Offload 2大系统系统瓶颈

- 部分 NPU 计算和存储 offload 到 CPU && DDR，涉及 CPU2NPU 间通信增加，如何避免 host2device 通信成为瓶颈？
- NPU 计算效率相比于 CPU 是数量级上的优，如何避免 CPU 参与过多计算，计算负载尽可能在 NPU？

ZeRO-Offload 原理

- 按计算量进行区分：
  - 高计算：forward && backward 计算量高，相关的权重参数计算 & 激活值计算仍然在 GPU；
  - 低计算：update 部分计算量低以通信为主，且需要的显存较大，放入 CPU & DDR。相关部分 Optimizer States（fp32）Update 和 Gradients (fp16) Update 等。



2.5 DeepSpeed

使用方式

多种使用方式

1. 使用 DeepSpeed 命令行工具运行训练脚本
2. HuggingFace 中 Transformers 通过 Trainer集成 Accelerator && DeepSpeed

多种使用方式

- 使用 DeepSpeed 命令行工具运行训练脚本

1. 在训练脚本中导入 DeepSpeed 模块
2. 在训练脚本中导入 Trainer 模块
3. 创建 Trainer 对象，将模型、训练数据集、优化器等参数传入
4. 使用 DeepSpeed 命令行工具运行训练脚本

```
deepspeed --num_gpus=8 train.py
```

```

deepspeed  --hostfile=hostfile  --master_port 60000 --include="node1:0,1,2,3@node2:0,1,2,3" run.py \
--deepspeed ds_config.json
```

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