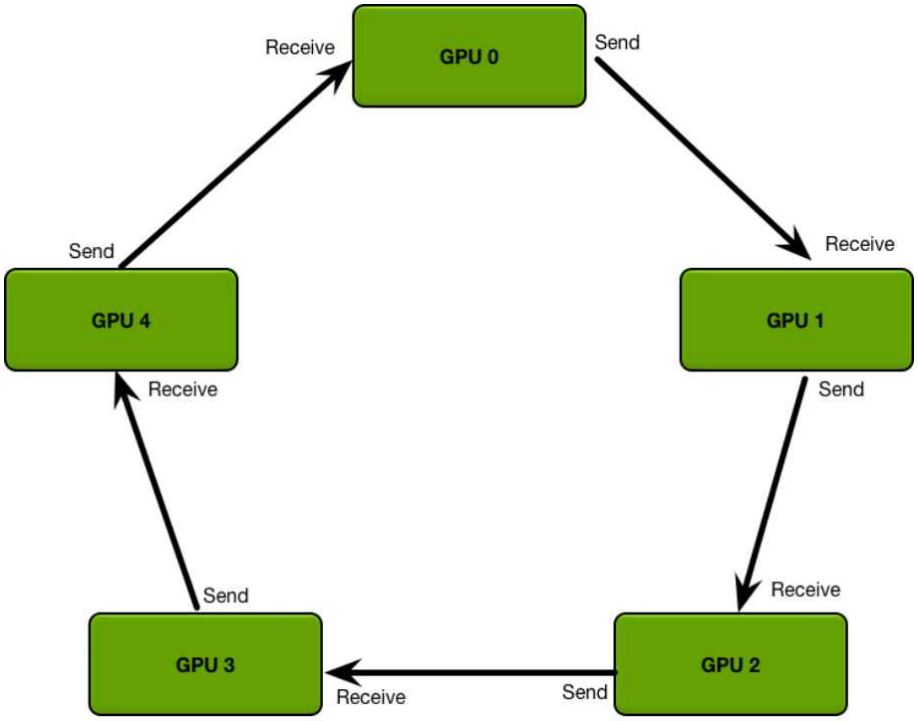
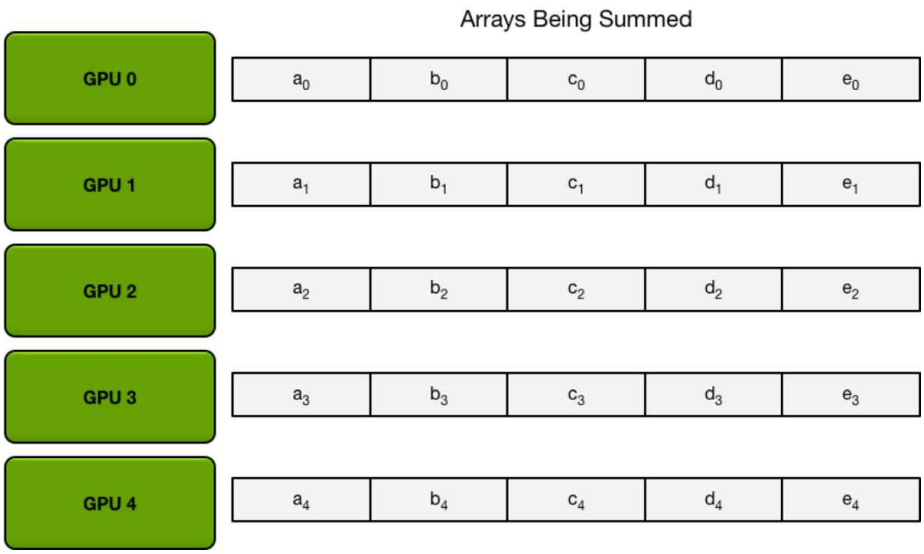
大模型 DeepSpeed 配置详解

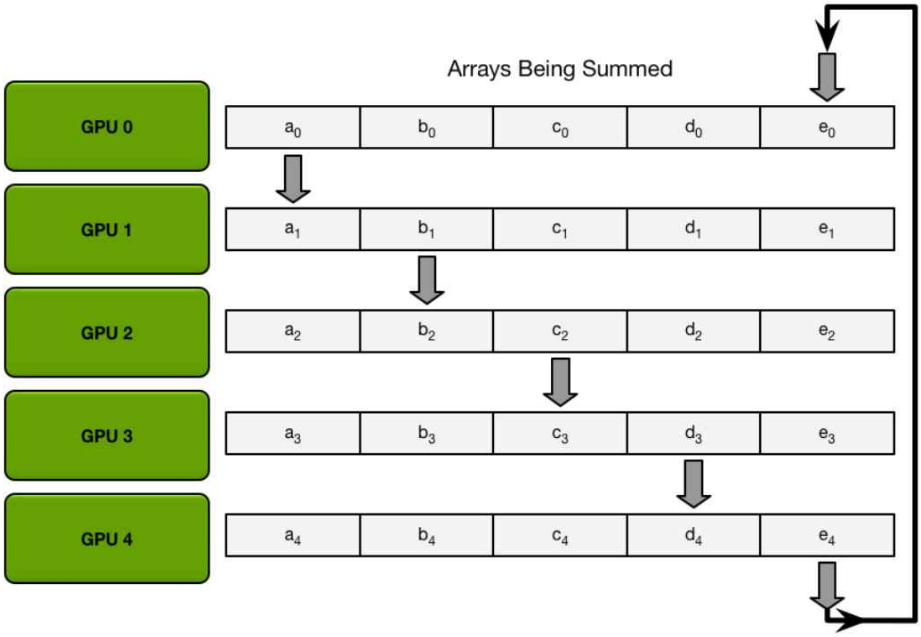
1. Stage2

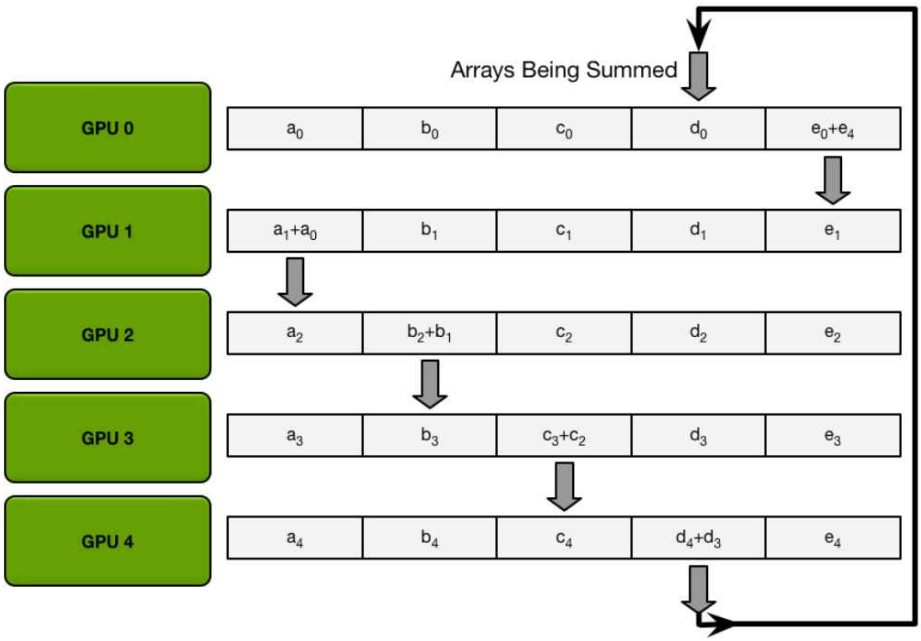
【Ring AllReduce】

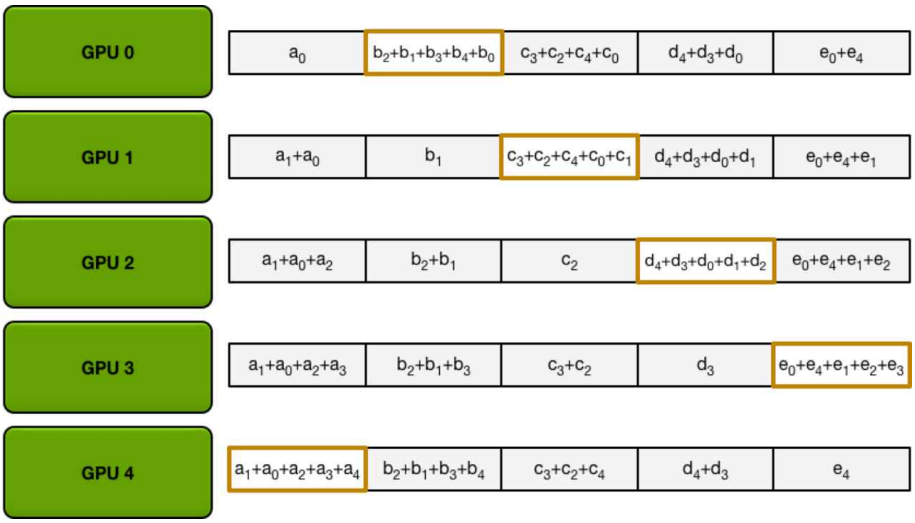


【Scatter-Reduce】

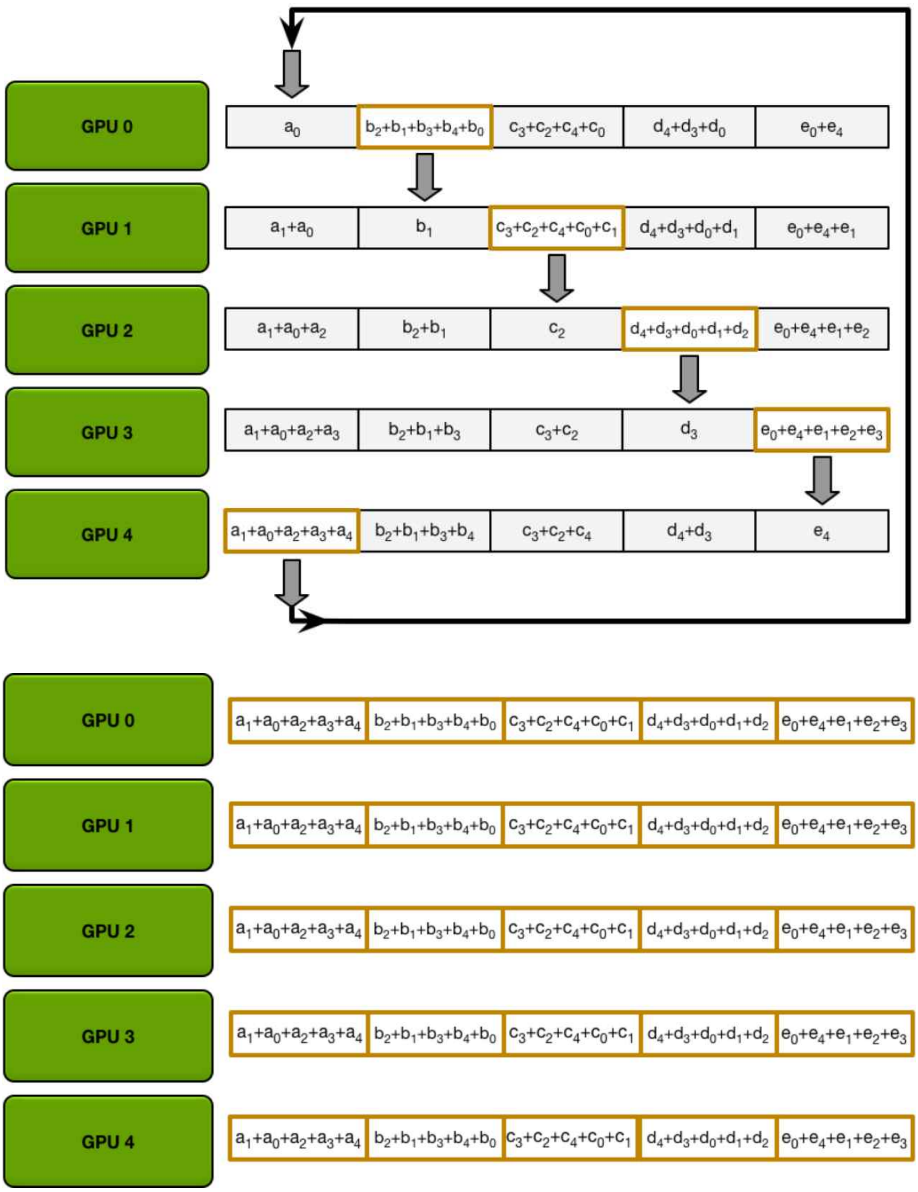








【The Allgather】




```
{
    "optimizer": {
      "type": "AdamW",
      "params": {
          "lr": "auto",
          "weight_decay": "auto",
          "torch_adam": true,
          "adam_w_mode": true
      }
    },
    "scheduler": {
      "type": "WarmupDecayLR",
      "params": {
          "warmup_min_lr": "auto",
          "warmup_max_lr": "auto",
          "warmup_num_steps": "auto",
          "total_num_steps": "auto"
      }
```

```
    },
    "fp16": {
        "enabled": true,
        "loss_scale": 0,
        "loss_scale_window": 1000,
        "initial_scale_power": 16,
        "hysteresis": 2,
        "min_loss_scale": 1
    },
    "zero_optimization": {
        "stage": 2,
        "allgather_partitions": true,
        "allgather_bucket_size": 2e8,
        "reduce_scatter": true,
        "reduce_bucket_size": "auto",
        "overlap_comm": true,
        "contiguous_gradients": true
    },
    "gradient_accumulation_steps": "auto",
    "gradient_clipping": "auto",
    "steps_per_print": 1000,
    "train_batch_size": "auto",
    "train_micro_batch_size_per_gpu": "auto",
    "wall_clock_breakdown": false
}
```

```
{
    "optimizer": {
      # 优化器类型
      "type": "AdamW",
      # 优化器参数
      "params": {
          # 学习率
          "lr": "auto",
          # 学习率衰减
          "weight_decay": "auto",
          "torch_adam": true,
          "adam_w_mode": true
      }
    },
    # LR scheduler
    "scheduler": {
      "type": "WarmupDecayLR",
      "params": {
          "warmup_min_lr": "auto",
```

```
            "warmup_max_lr": "auto",
            "warmup_num_steps": "auto",
            "total_num_steps": "auto"
        }
    },
    # 使用利用 NVIDIA 的 Apex 包的 mixed precision/FP16 训练的配置
    # 该模式类似于 AMP 的 O2 模式
    # O0：纯FP32训练，可以作为accuracy的baseline。
    # O1：混合精度训练（推荐使用），根据黑白名单自动决定使用FP16（GEMM，卷积）还是FP32（Softmax）进行计算。
    # O2：“几乎FP16”混合精度训练，不存在黑白名单，除了Batch norm，几乎都是用FP16计算。
    # O3：纯FP16训练，很不稳定，但是可以作为speed的baseline；
    "fp16": {
        # 是否启用fp16训练
        "enabled": true,
        # fp16训练的损失缩放值
        # 反向计算起始，Loss首先乘以Loss Scale值进行扩大，避免反向梯度过小而产生下溢
        # 这样在反向传播过程中所有值都扩大了相同倍数
        # 设置为0.0时为 dynamic loss scaling
        # 设置为其他值将会使用 static fixed loss scaling
        "loss_scale": 0,
        # 升高/降低动态损失比例值的窗口
        "loss_scale_window": 1000,
        # 初始 dynamic loss scale 的幂参数（2^initial_scale_power）
        "initial_scale_power": 16,
        # 表示动态损耗缩放中的延迟偏移
        "hysteresis": 2,
        # 最小的 dynamic loss scale 数值
        "min_loss_scale": 1
    },
    # 启用和配置 ZERO 优化，与 FP16/BF16/FP32 和 Adam 优化器兼容。
    "zero_optimization": {
        # 选择 ZeRO 优化器的不同阶段。 Stage 0、1、2、3分别指禁用、优化器状态分区、优化器+梯度状态分区、优化器+梯度+参数分区。
        "stage": 2,
        # 是否使用 reduce-scatter 算法来进行梯度聚合。
        # Reduce-scatter 是一种并行计算中常用的算法，可以将数据均匀地分发给各个节点，并同时进行聚合操作，从而实现高效的梯度聚合。这种算法可以减少全局通信的开销，并提高训练的性能。
        "reduce_scatter": true,
        # 一次 reduce 的数据量
        "reduce_bucket_size": "auto",
        # 使用 allgather 在每个步骤结束时从所有 GPU 收集更新的参数
        "allgather_partitions": true,
        # 一次 allgather 的数据量
        "allgather_bucket_size": "auto",
        # 尝试将梯度的更新与反向传播重叠进行
```

```
        "overlap_comm": true,
        # 在生成梯度时将梯度复制到连续缓冲区。实现高效访问数据，避免反向传播期间的内存碎片。
        # 通过设置 contiguous_gradients 参数为 true，DeepSpeed 可以要求梯度在内存中是连续存储的。这样做可以确保梯度聚合操作能够高效地进行，减少了数据拷贝和内存访问的开销。
        # 相反，当 contiguous_gradients 设置为 false 时，DeepSpeed 不要求梯度在内存中是连续存储的。这意味着梯度聚合操作可能需要进行额外的数据拷贝和内存访问，导致性能下降。
        "contiguous_gradients": true
    },
    # 梯度累计步数
    # 在平均和使用梯度之前对梯度进行累计的步数，
    # ① 会导致步骤之间的梯度通信频率降低 ② 能够使用每个 GPU 进行更大的批量训练
    "gradient_accumulation_steps": "auto",
    # 梯度裁剪，避免梯度爆炸，对超出预设范围的梯度值，进行梯度裁剪
    "gradient_clipping": "auto",
    # 信息打印间隔
    "steps_per_print": 1000,
    # 训练 batch_size
    # train_batch_size must be equal to train_micro_batch_size_per_gpu * gradient_accumulation * number of GPUs，3个参数指定2/3即可
    "train_batch_size": "auto",
    # 每个 GPU 的 micro_batch_size
    "train_micro_batch_size_per_gpu": "auto",
    # DeepSpeed 会在训练结束时输出一个时间统计报告。该报告显示了整个训练过程中的各个步骤的耗时情况，以及每个步骤中的细分时间。
    # 这些步骤可能包括数据加载、前向传播、反向传播、优化器更新等等。通过查看时间统计报告，你可以了解每个步骤所花费的时间，并且可以根据需要进行性能优化和调整。
    "wall_clock_breakdown": false
}
```

2. Stage3

```
{
    "fp16": {
        "enabled": true,
        "auto_cast": false,
        "loss_scale": 0,
        "initial_scale_power": 32,
        "loss_scale_window": 1000,
        "hysteresis": 2,
        "min_loss_scale": 1
    },
    "optimizer": {
        "type": "AdamW",
        "params": {
```

```
            "lr": "auto",
            "betas": [
              0.9,
              0.999
            ],
            "eps": 1e-8,
            "weight_decay": "auto"
        }
    },
    "zero_optimization": {
        "stage": 3,
        "overlap_comm": true,
        "contiguous_gradients": true,
        "stage3_prefetch_bucket_size": "auto",
        "stage3_param_persistence_threshold": "auto",
        "stage3_max_live_parameters": 0,
        "stage3_max_reuse_distance": 0,
        "stage3_gather_16bit_weights_on_model_save": true
    },
    "train_batch_size": "auto",
    "train_micro_batch_size_per_gpu": "auto",
    "wall_clock_breakdown": false
}
```

```
{
    "fp16": {
        "enabled": true,
        "auto_cast": false,
        "loss_scale": 0,
        "initial_scale_power": 32,
        "loss_scale_window": 1000,
        "hysteresis": 2,
        "min_loss_scale": 1
    },
    "optimizer": {
        "type": "AdamW",
        "params": {
            "lr": "auto",
            "betas": [
              0.9,
              0.999
            ],
            "eps": 1e-8,
            "weight_decay": "auto"
        }
```

```
    },
    "zero_optimization": {
        "stage": 3,
        # 启用将优化器状态卸载到 CPU 或 NVMe，并将优化器计算卸载到 CPU。 这可以为更大的模型或批量大小释放 GPU 内存。 适用于 ZeRO 阶段 1、2、3。
        "offload_optimizer": {
            "device": "cpu",
            "pin_memory": true
        },
        # 启用和配置参数卸载到 CPU 或 NVMe，仅适用于 ZeRO stage 3。请注意，如果未指定或不支持“device”的值，则会触发异常。
        "offload_param": {
            "device": "cpu",
            "pin_memory": true,
            "max_in_cpu": 1.4e10
        },
        # 尝试将梯度的更新与反向传播重叠进行
        "overlap_comm": true,
        # 在生成梯度时将其复制到连续的缓冲区，避免向后传递期间出现内存碎片。
        "contiguous_gradients": true,
        # 用于预取参数的固定缓冲区的大小
        # 较小的值使用较少的内存，但可能会因通信而增加停顿。
        "stage3_prefetch_bucket_size": "auto",
        # 不对小于这个阈值的参数进行partition
        # 较小的数值使用的内存更少，但可能会极大地增加通讯负担
        "stage3_param_persistence_threshold": "auto",
        # 每个 GPU 驻留的最大参数量
        # 较小的数值使用的内存更少，但需要进行更多的通讯
        "stage3_max_live_parameters": 0,
        # 如果参数在这个阈值范围内会被复用，不会被释放
        # 较小的数值使用的内存更少，但需要进行更多的通讯
        "stage3_max_reuse_distance": 0,
        # 在通过 save_16bit_model() 保存模型之前合并权重
        # 由于权重在 GPU 之间进行 partition，启用此选项时该函数会自动收集权重，然后保存 fp16 模型权重。
        "stage3_gather_16bit_weights_on_model_save": true
    },
    "train_batch_size": "auto",
    "train_micro_batch_size_per_gpu": "auto",
    "wall_clock_breakdown": false
}
```