PR #33667 完整报告

sgl-project/sglang

[diffusion] Pack Ulysses Q/K/V input all-to-all into one collective + reusable a2a staging buffers

合并时间：2026-08-05 19:15

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/33667

执行摘要

- 一句话：Ulysses QKV all-to-all 打包为一次集合通信并复用暂存缓冲区
- 推荐动作：值得精读，特别是 _a2a_staging_buffer 的绕过条件设计和 _usp_input_all_to_all_qkv 的 fallback 策略。它展示了如何在保证位精确的前提下削减集合通信次数与分配开销，并用单卡模拟多 rank 的测试手法验证位一致性。对关注 diffusion 推理性能、Ulysses 并行或集合通信优化的工程师是很好的参考案例。

功能与动机

PR body 指出，在 Ulysses 序列并行下每个 USP 注意力层都要承担输入交换（sequence-sharded -> head-sharded）和输出交换，通用路径的输入交换目前是 3 次 all_to_all_single（Q/K/V 各一次），每次还要在发送侧做 permute + contiguous、接收侧做 5D permute + contiguous，而且每个集合通信在每层每个 denoise step 都新建 send/recv 缓冲区。Profiling LingBot-World realtime 显示 4-GPU 下 Ulysses a2a 链路约 30% 的暴露开销。

实现拆解

1. 新增可复用暂存缓冲区 _a2a_staging_buffer（usp.py）：按 (role, shape, dtype, device) 缓存发送 / 接收缓冲区；在 autograd、torch.compile、CUDA 图捕获或非 CUDA 设备上绕过缓存，避免图私有内存池与 eager 重放共享。_usp_all_to_all_single 增加可选 role 参数，role=None 时保持原有的每次新建行为。
2. 新增打包 Q/K/V 输入交换 _usp_input_all_to_all_qkv（usp.py）：对 heads at dim=2 的 4D 输入，先通过 _can_use_packed_qkv_a2a_4d 检查 eligibility（CUDA、bf16/fp16、contiguous、shape 可整除 world_size、非编译），再调用 pack_qkv_destination_major 一次写入 Q/K/V 到暂存发送缓冲区，用 1 次 all_to_all_single 完成交换，接收侧用 view 或一次 permute+contiguous 解开，最终 split 并拷贝出。不满足条件的输入回退到原有三次 _usp_input_all_to_all 调用。
3. 扩展 Triton 内核 pack_qkv_destination_major（ulysses_qkv.py）：新增可选 out 参数，写入前断言 shape/dtype/contiguity，使打包结果可直接落入暂存缓冲区。
4. 接入通用注意力路径（attention/layer.py）：两处通用 USPAttention 输入交换调用点由三次单独 exchange 替换为一次 _usp_input_all_to_all_qkv；H3 路径的 _usp_input_all_to_all_packed_qkv 与 _usp_output_all_to_all 也改为使用复用缓冲区。

5. 测试配套：新增 test_usp_packed_qkv_a2a.py 单测，用单卡模拟 4/2 rank 的 all_to_all_single 分块语义，验证打包路径与数学 spec 位一致；test_minimax_h3_dit_contract.py 的 fake_all_to_all 增加 role=None 参数以适配新接口。

关键文件：

- python/sglang/multimodal_gen/runtime/layers/usp.py（模块 并行传输；类别 source；类型 core-logic；符号 _a2a_staging_buffer, _usp_all_to_all_single, _can_use_packed_qkv_a2a_4d, _usp_input_all_to_all_qkv）：核心逻辑所在：新增可复用 staging buffer、打包 Q/K/V 交换和 eligibility 判定，是本次 PR 性能优化的主要承载文件。
- python/sglang/multimodal_gen/test/unit/test_usp_packed_qkv_a2a.py（模块 单元测试；类别 test；类型 test-coverage；符号 TestPackedQKVInputA2A, _run_all_ranks, fake_a2a, test_packed_matches_unpacked_bitwise）：新增单测，用单卡模拟 4/2 rank 的 all_to_all_single 语义，验证打包路径与数学 spec 位一致，是本次优化正确性的核心保障。
- python/sglang/multimodal_gen/runtime/layers/attention/layer.py（模块 注意力层；类别 source；类型 core-logic）：两处通用 USPAttention 输入交换调用点切换到打包路径，是优化生效的入口。
- python/sglang/kernels/ops/diffusion/triton/ulysses_qkv.py（模块 内核适配；类别 infra；类型 infrastructure）：Triton pack 内核支持预分配 out 缓冲区，让打包结果直接写入暂存区，避免额外分配。
- python/sglang/multimodal_gen/test/unit/test_minimax_h3_dit_contract.py（模块 契约测试；类别 test；类型 test-coverage；符号 fake_all_to_all）：既有 H3 契约测试的 fake_collective 需要接收新增的 role 参数以匹配新接口。

关键符号：_a2a_staging_buffer, _usp_input_all_to_all_qkv, _can_use_packed_qkv_a2a_4d, _usp_all_to_all_single, _usp_input_all_to_all_packed_qkv, _usp_output_all_to_all

关键源码片段

python/sglang/multimodal_gen/runtime/layers/usp.py

核心逻辑所在：新增可复用 staging buffer、打包 Q/K/V 交换和 eligibility 判定，是本次 PR 性能优化的主要承载文件。

```
# python/sglang/multimodal_gen/runtime/layers/usp.py（节选）

# 模块级缓存：按 (role, shape, dtype, device) 复用 Ulysses 集合通信的暂存缓冲区
_A2A_STAGING_BUFFERS: dict[tuple, torch.Tensor] = {}

def _a2a_staging_buffer(
    role: str, shape: tuple[int, ...], dtype: torch.dtype, device: torch.device
) -> torch.Tensor:
    """为 Ulysses 集合通信获取可复用暂存缓冲区。

    同一 role 的缓冲区在流序上会被完全消费后才被下一次同 role 集合覆盖，
    因此按 (role, shape, dtype, device) 缓存是精确的，可消除分配器抖动。
```

```
    在 autograd、torch.compile 或 CUDA 图捕获时绕过：若缓冲区在捕获期间
    首次分配，会落入图私有内存池，不能与 eager 重放共享。
    """
    if (
        torch.is_grad_enabled()
        or torch.compiler.is_compiling()
        or device.type != "cuda"
        or torch.cuda.is_current_stream_capturing()
    ):
        # 以上场景无法安全复用，回退为每次新建
        return torch.empty(shape, dtype=dtype, device=device)
    key = (role, tuple(shape), dtype, device.index)
    buffer = _A2A_STAGING_BUFFERS.get(key)
    if buffer is None:
        buffer = torch.empty(shape, dtype=dtype, device=device)
        _A2A_STAGING_BUFFERS[key] = buffer
    return buffer


def _usp_input_all_to_all_qkv(
    q: torch.Tensor, k: torch.Tensor, v: torch.Tensor
) -> tuple[torch.Tensor, torch.Tensor, torch.Tensor]:
    """Ulysses 输入交换：头部在 dim=2 的 Q/K/V 打包为一次集合通信。

    [b, s_local, h, d] x3 -> [b, s_global, h_local, d] x3。
    仅改变数据搬运方式，结果与未打包路径位一致；不满足条件的输入回退。
    """
    world_size = get_ulysses_parallel_world_size()
    if world_size <= 1:
        return q, k, v
    if not _can_use_packed_qkv_a2a_4d(q, k, v, world_size):
        # 回退：CPU、GQA 形状不匹配、非连续、fp32、编译中等场景
        return (
            _usp_input_all_to_all(q, head_dim=2),
            _usp_input_all_to_all(k, head_dim=2),
            _usp_input_all_to_all(v, head_dim=2),
        )

    b, s_local, h_global, d = q.shape
    h_local = h_global // world_size
    rows = b * s_local
    # 一个 Triton 内核按 destination-major 布局直接写 Q/K/V 到暂存区，
    # 省去发送侧 3 次 permute + contiguous 预拷贝
    packed = pack_qkv_destination_major(
        q.view(rows, h_global, d),
        k.view(rows, h_global, d),
        v.view(rows, h_global, d),
        world_size,
        out=_a2a_staging_buffer(
```

```
            "usp_packed_qkv_src", (world_size, rows, h_local, 3 * d), q.dtype, q.device
        ),
    )
    # 单个 all_to_all_single 完成三个张量的交换，接收侧写入复用缓冲
    packed = _usp_all_to_all_single(packed, role="usp_packed_qkv_recv")
    if b == 1:
        # 收到的分块已经是 sequence-major：rank j 的数据位于偏移 j * s_local，
        # 直接 view 即可，无需真正 transpose
        packed = packed.view(1, world_size * s_local, h_local, 3 * d)
    else:
        packed = (
            packed.view(world_size, b, s_local, h_local, 3 * d)
            .permute(1, 0, 2, 3, 4)
            .contiguous()
            .view(b, world_size * s_local, h_local, 3 * d)
        )
    q, k, v = packed.split(d, dim=-1)
    # 在暂存缓冲区被下一次集合覆盖前拷贝出来
    return q.contiguous(), k.contiguous(), v.contiguous()
```

python/sglang/multimodal_gen/test/unit/test_usp_packed_qkv_a2a.py

新增单测，用单卡模拟 4/2 rank 的 all_to_all_single 语义，验证打包路径与数学 spec 位一致，是本次优化正确性的核心保障。

```
# python/sglang/multimodal_gen/test/unit/test_usp_packed_qkv_a2a.py（节选）
# 单卡模拟 4/2 rank 的 all_to_all_single 分块语义，验证打包路径与数学 spec 位一致

@unittest.skipUnless(torch.cuda.is_available(), "requires CUDA")
class TestPackedQKVInputA2A(unittest.TestCase):
    def _run_all_ranks(self, fn, world):
        sends, recvs = [], None

        def fake_a2a(x, role=None):
            # 第一遍记录每 rank 的发送数据，第二遍按 chunk 语义回放
            if recvs is None:
                sends.append(x.detach().clone())
                return torch.empty_like(x)
            return recvs.pop(0).reshape(x.shape)

        with (
            patch(f"{_USP}._usp_all_to_all_single", fake_a2a),
            patch(f"{_USP}.get_ulysses_parallel_world_size", return_value=world),
        ):
            for r in range(world):
                fn(r)
            # 构造正确语义：rank r 发给 rank j 的第 j 个 chunk 成为 rank j 收到的第 r 个 chunk
            recvs = [
                torch.cat([s.flatten().chunk(world)[r] for s in sends])
                for r in range(world)
```

```
        ]
        return [fn(r) for r in range(world)]

    def test_packed_matches_unpacked_bitwise(self):
        # 覆盖 world=4、batch=1 和 world=2、batch=2 两种配置
        for world, b, s_global, h_global, d in ((4, 1, 128, 8, 64), (2, 2, 48, 6, 32)):
            torch.manual_seed(1234)
            s_local, h_local = s_global // world, h_global // world
            full = [
                torch.randn(b, s_global, h_global, d, dtype=torch.bfloat16, device="cuda")
                for _ in range(3)
            ]
            shards = [
                tuple(t[:, r * s_local : (r + 1) * s_local].contiguous() for t in full)
                for r in range(world)
            ]
            packed = self._run_all_ranks(
                lambda r: usp_mod._usp_input_all_to_all_qkv(*shards[r]), world
            )
            for r in range(world):
                for i in range(3):
                    # 数学 spec：全局序列拼接后取第 r 个 head 分片
                    spec = full[i][:, :, r * h_local : (r + 1) * h_local].contiguous()
                    self.assertTrue(torch.equal(packed[r][i], spec), f"rank{r} qkv[{i}]")
                    self.assertTrue(packed[r][i].is_contiguous())
```

评论区精华

本 PR 没有 review 评论。设计权衡在 PR body 中自述：staging buffer 在 autograd、torch.compile、CUDA 图捕获下必须绕过，否则捕获期间分配的缓冲区会落入图私有内存池，无法与 *eager* 重放安全共享；不满足 pack 条件的输入（CPU、GQA 形状不匹配、非连续、fp32、编译中）回退到原有 unpacked 路径，保证位精确。作者还对比了 async3 方案（三个 unpacked 集合并发排队后等待），实测不如单次 packed 集合，因此未采纳。

- 暂无高价值评论线程

风险与影响

- 风险：
  - 缓冲区复用正确性：*_a2a_staging_buffer* 依赖流序保证同一 role 的缓冲区被完全消费后才覆盖；若未来引入异步多流或乱序执行，可能发生读写竞争。当前实现仅在同步等待场景下使用。
  - CUDA 图兼容性：函数在 is_current_stream_capturing() 时绕过缓存，但若同一进程在 *capture* 前已缓存 buffer，*eager* 阶段复用时可能污染捕获结果；目前通过明确绕过规避。
  - 通用路径改造波及面：layer.py 两处调用点替换影响所有走普通 USPAttention 的 diffusion 模型（Wan 2.x、H3 等），虽然有 fallback，但新路径的任何 bug 都会扩散。

- 位精确性依赖：性能收益建立在纯数据搬运、逐位一致的前提上；一旦内核或视图逻辑改变，需要重新验证。
- 收益范围有限：仅对 Ulysses world size > 1 且 heads at dim=2 的模型生效，单卡或 head at dim=1 的路径不受影响。
- 影响：影响范围集中在 multimodal_gen 运行时：核心改动在 usp.py 与 attention/layer.py，并涉及 Triton 内核与两个测试文件。受益用户为使用 Wan 2.x、MiniMax-H3 等 diffusion 模型进行多 GPU Ulysses 推理的场景，输入交换阶段提速 1.28-1.31x，端到端约 -1.7% 至 -2.1%。系统层面减少了每次 denoise step 的集合通信次数和临时分配次数，缓解 allocator 抖动。团队层面沉淀了“staging buffer 复用 + 打包集合通信”的可复用性能优化模式，以及单卡模拟多 rank 集合语义的测试方法论，为后续其他并行通信优化提供了参考。
- 风险标记：缓存复用正确性，CUDA 图 /compile 兼容性，通用路径改造波及多模型，位精确性依赖

关联脉络

- PR #33275 Port sol-engine Ulysses relayout kernels: PR #33667 在 body 中明确提到 #33275 已移植 pack_qkv_destination_major 与 usp_merge_heads 内核，本 PR 在其基础上推广到通用 USP 路径。