PR #35538 完整报告

sgl-project/sglang

[diffusion] fix: stop reserving NCCL device buffers for single-rank groups

合并时间：2026-08-20 09:04

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35538

执行摘要

- 一句话：单卡 diffusion 省约 5 GiB：单 rank 组不再建 NCCL
- 推荐动作：值得精读。核心设计决策有两点：一是用 backend 选择而非跳过创建来根治 NCCL 的显存预留，因为 torch.distributed.new_group 本身就会触发分配；二是用统一工厂函数收敛 9 处创建点，避免逐个调用点手动判断、漏改。cache-dit 的补丁也展示了“改后端后要重新审视所有裸消费点”的完整思路。阅读时可重点关注 GroupCoordinator 的 world_size == 1 短路机制与 cache_dit_integration.py 的防护逻辑，后续新增并行组时统一走 new_device_group。

功能与动机

PR body 指出："On a single GPU, ~5 GiB of VRAM is gone before a single weight is loaded. On a 12 GiB consumer card that is 44% of the device, and it makes VAE decode fail at any resolution.根因是单卡运行仍构建全部并行组（DP、CFG、TP、SP、PP、VAE_DECODE、DIT、VAE、ulysses、ring），每组单 rank 都建 NCCL 进程组，NCCL 在创建 communicator 时按 channel 预留设备 buffer：world 组约 470 MiB、其余每组约 390 MiB，而这些 buffer 永远不可能被使用——GroupCoordinator 所有 collective 在 world_size == 1 时提前返回，device communicator 只在 world_size > 1 时构建，pynccl 也已跳过单 rank 组，唯独 torch.distributed.new_group 没有。NCCL_MAX_NCHANNELS=1 可把每组成本降到 10 MiB，进一步证实是 per-channel buffer 的浪费。

实现拆解

1. 在 python/sglang/multimodal_gen/runtime/distributed/group_coordinator.py 新增统一工厂函数 new_device_group(ranks, backend=None)：单 rank 组强制使用 gloo backend，多 rank 组原样透传请求的 backend（None 时由 torch.distributed 决定默认值）。这样从源头避免 NCCL 为永不通信的单 rank 组预留 per-channel 设备 buffer，同时保持多卡语义不变。
2. 替换 9 处设备组创建点：GroupCoordinator.__init__的 device_group 创建、PipelineGroupCoordinator 单 rank 分支及其 skip_device_group；parallel_state.py 的 init_dit_group 与 init_vae_group；parallel_groups.py 的 set_seq_parallel_pg_by_sp_groups 中 4 个 ulysses/ring 组。两 rank 的 device_group_0_1 / device_group_1_0 pipeline 对保持不变，因为它们只在恰好两 rank 时可达。

3. 补齐隐藏消费点防护：cache_dit_integration.py 的 patched_similarity 在 target_group 为 None 或 dist.get_world_size(target_group) == 1 时直接返回原始 similarity，避免单卡上 DIT 组（现为 gloo）的 all-reduce 每次都要把设备张量经 host 搬运——单 rank 求平均本就等于原样返回输入。
4. 测试与验证配套：新增 test_single_rank_device_group.py（3 个测试、5 个 subtest），用 mock 断言 torch.distributed.new_group 的 backend 选择；现有 test_cache_dit_integration.py 12 个测试通过；RTX 3060 上输出 SHA-256 与 NCCL_MAX_NCHANNELS=1 对照组一致（bit-exact），两卡回归输出一致；CI 除预先存在的 NPU lane 故障外全部通过。

关键文件：

- python/sglang/multimodal_gen/runtime/distributed/group_coordinator.py（模块 通信组；类别 source；类型 core-logic；符号 new_device_group）：核心改动文件：新增 new_device_group() 工厂，单 rank 组强制 gloo，并替换 GroupCoordinator.init、PipelineGroupCoordinator 单 rank 分支与 skip_device_group 共 3 处创建点，是本次显存修复的根源。
- python/sglang/multimodal_gen/runtime/distributed/parallel_groups.py（模块 并行组；类别 source；类型 dependency-wiring；符号 set_seq_parallel_pg_by_sp_groups）：ulysses 与 ring 共 4 个序列并行组的创建点全部改走 new_device_group，并移除对 torch.distributed 的直接 import，是 9 处路由中覆盖组数最多的文件。
- python/sglang/multimodal_gen/runtime/distributed/parallel_state.py（模块 并行状态；类别 source；类型 core-logic；符号 init_dit_group, init_vae_group）：init_dit_group 与 init_vae_group 改用 new_device_group，覆盖 DIT 与 VAE 两个独立进程组，单卡时不再各自预留约 390 MiB。
- python/sglang/multimodal_gen/runtime/cache/cache_dit_integration.py（模块 缓存集成；类别 source；类型 core-logic；符号 patched_similarity）：本次改动发现的唯一隐藏消费点：DIT 组改为 gloo 后，单卡上残留的 all-reduce 会每次经 host 搬运张量，必须加 world_size == 1 短路。
- python/sglang/multimodal_gen/test/unit/test_single_rank_device_group.py（模块 单元测试；类别 test；类型 test-coverage；符号 TestSingleRankDeviceGroup, test_single_rank_group_avoids_the_device_backend, test_multi_rank_group_keeps_the_requested_backend, test_backend_defaults_to_none_for_multi_rank）：新增单元测试，以 mock 方式锁定 new_device_group 的 backend 选择规则：单 rank 强制 gloo、多 rank 保留请求 backend、缺省透传 None，共 3 测试 5 subtest。

关键符号：new_device_group, init_dit_group, init_vae_group, set_seq_parallel_pg_by_sp_groups, patched_similarity

关键源码片段

python/sglang/multimodal_gen/runtime/distributed/group_coordinator.py

核心改动文件：新增 new_device_group() 工厂，单 rank 组强制 gloo，并替换 GroupCoordinator.init、PipelineGroupCoordinator 单 rank 分支与 skip_device_group 共 3 处创建点，是本次显存修复的根源。

```
def new_device_group(ranks, backend=None):
    """Create a process group for device collectives.

    A single-rank group never runs one: every collective short-circuits on
    world_size == 1. NCCL would still allocate its per-channel device buffers
    for it, which costs ~390 MiB a group.
    """
    # 单 rank 组永远不会发生真正的设备集合通信：GroupCoordinator 内部所有
    # collective 都在 world_size == 1 时提前返回（"Bypass the function if we
    # are using only 1 GPU"）。但 torch.distributed.new_group 不理会这一点，
    # 仍会为 NCCL 的每个 channel 预留设备 buffer，实测每个单 rank 组约
    # 388 MiB、world 组约 470 MiB，且永远不会被用到。这里强制退回 gloo，
    # 从根上避免预留；多 rank 组则原样透传请求的 backend（None 表示默认）。
    return torch.distributed.new_group(
        ranks, backend="gloo" if len(ranks) == 1 else backend
    )


class GroupCoordinator:
    def __init__(
        self,
        group_ranks: List[List[int]],
        local_rank: int,
        torch_distributed_backend: Union[str, Backend],
        use_device_communicator: bool = True,
        use_srt_custom_allreduce: bool = False,
        use_message_queue_broadcaster: bool = False,
        group_name: str | None = None,
    ):
        self.unique_name = _get_unique_name(group_name)
        self.rank = torch.distributed.get_rank()
        self.local_rank = local_rank
        self.device_group = None
        self.cpu_group = None

        for ranks in group_ranks:
            # 统一走 new_device_group：单卡时 DP/CFG/TP/SP/PP/VAE_DECODE 等
            # 所有并行轴都是 [0] 单 rank 组，由工厂自动降级 gloo，逐个省下
            # 约 390 MiB；多卡时保持原有 NCCL 语义不变。
            device_group = new_device_group(ranks, torch_distributed_backend)
            # CPU 协调组仍固定用 gloo，与本改动无关，行为保持不变。
            with suppress_stdout():
                cpu_group = torch.distributed.new_group(ranks, backend="gloo")
            if self.rank in ranks:
                self.ranks = ranks
```

```
        self.world_size = len(ranks)
        self.rank_in_group = ranks.index(self.rank)
        self.device_group = device_group
        self.cpu_group = cpu_group
```

python/sglang/multimodal_gen/runtime/distributed/parallel_state.py

init_dit_group 与 init_vae_group 改用 new_device_group，覆盖 DIT 与 VAE 两个独立进程组，单卡时不再各自预留约 390 MiB。

```
def init_dit_group(dit_parallel_size: int, backend: str) -> None:
    global _DIT
    assert _DIT is None, "DIT group is already initialized"
    # 单卡时 dit_parallel_size == 1：new_device_group 自动退回 gloo，
    # 不再为这条永远不通信的 DIT 组预留 NCCL 设备 buffer（约 390 MiB）。
    _DIT = new_device_group(list(range(dit_parallel_size)), backend)


def init_vae_group(dit_parallel_size: int, vae_parallel_size: int, backend: str):
    # Initialize VAE group first
    global _VAE
    assert _VAE is None, "VAE parallel group is already initialized"
    vae_ranks = list(range(dit_parallel_size, dit_parallel_size + vae_parallel_size))
    # 与 DIT 组同理：单 rank 的 VAE 组也走 gloo 路径，省下同样的显存。
    _VAE = new_device_group(vae_ranks, backend)
```

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

新增单元测试，以 mock 方式锁定 new_device_group 的 backend 选择规则：单 rank 强制 gloo、多 rank 保留请求 backend、缺省透传 None，共 3 测试 5 subtest。

```
"""Single-rank groups get gloo, so NCCL does not reserve device buffers for them."""

import unittest
from unittest.mock import patch

from sglang.multimodal_gen.runtime.distributed.group_coordinator import (
    new_device_group,
)

NEW_GROUP_PATH = "torch.distributed.new_group"


class TestSingleRankDeviceGroup(unittest.TestCase):
    def test_single_rank_group_avoids_the_device_backend(self):
        # 单 rank 组无论请求 nccl / hccl 还是 None，都必须落到 gloo。
        for ranks, requested in [([0], "nccl"), ([3], "hccl"), ([0], None)]:
            with self.subTest(ranks=ranks, requested=requested):
                with patch(NEW_GROUP_PATH) as new_group:
                    new_device_group(ranks, requested)
                new_group.assert_called_once_with(ranks, backend="gloo")
```

```
    def test_multi_rank_group_keeps_the_requested_backend(self):
        # 多 rank 组保留原 backend，保证多卡路径语义不变。
        for ranks, requested in [([0, 1], "nccl"), ([0, 1, 2, 3], None)]:
            with self.subTest(ranks=ranks, requested=requested):
                with patch(NEW_GROUP_PATH) as new_group:
                    new_device_group(ranks, requested)
                new_group.assert_called_once_with(ranks, backend=requested)

    def test_backend_defaults_to_none_for_multi_rank(self):
        # backend 缺省时透传 None，由 torch.distributed 决定默认后端。
        with patch(NEW_GROUP_PATH) as new_group:
            new_device_group([0, 1])
        new_group.assert_called_once_with([0, 1], backend=None)


if __name__ == "__main__":
    unittest.main()
```

评论区精华

本 PR 没有正式 review 评论，核心论证集中在 PR body 与 issue 评论的 CI 归因中。PR body 用裸 torch.distributed 复现数据给出根因证据：CUDA context 114 MiB、加 world 组 584 MiB、每个额外 new_group([0]) 再 +388 MiB，并以 NCCL_MAX_NCHANNELS=1 对照组确认这些是永不使用的 per-channel buffer。作者还明确圈定所有已短路消费点（GroupCoordinator 的 world_size == 1 提前返回、usp.py 的 early return、attention/layer.py 的 sp_size > 1 门槛等），并主动发现并修复了 cache-dit 这一唯一的漏网消费点。CI 归因评论逐项说明所有受影响的 lane 全绿，NPU lane 的红测在不带本改动的分支上也同样失败（见 #35511），不属于本 PR 引入。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. backend 切换后的隐藏消费点风险：单 rank 组从 NCCL 换成 gloo 后，任何仍试图在单 rank 组上执行设备 collective 的代码都会从 GPU 通信退化为经 host 搬运，可能引入性能退化甚至报错。作者已排查 GroupCoordinator 短路逻辑、usp.py、attention/layer.py、VAE getter 等已知消费点，并修复了 cache-dit 这一漏网点，但仓库中仍可能存在其他裸 ProcessGroup 消费点未被覆盖，需依赖单卡 diffusion 的 CI 覆盖来兜底。此处风险具体落在 group_coordinator.py、parallel_state.py、cache_dit_integration.py 的调用路径上。
  2. 多 rank 行为一致性：backend=None 的透传与原先 ulysses/ring 组的创建行为一致；两 rank pipeline 组未改动，因此多卡路径不受影响，已由两卡回归（SHA-256 一致）验证。
  3. 对 NCCL 版本行为的依赖：节省的显存数值（约 390 MiB/ 组）依赖具体 NCCL 版本的 per-channel buffer 大小，不同版本收益可能浮动，但改动方向不变。 - 影响：对用户

：单卡 diffusion 显存敏感场景（尤其 12 GiB 消费卡）是最大受益方——加载后可用显存从 6.51 GiB 提升到 11.06 GiB，峰值显存从 11552 MiB 降到 6830 MiB，VAE 解码从任意分辨率 OOM 恢复为正常。多卡用户零感知，输出与改动前完全一致。对系统：新增了统一的设备组创建入口 new_device_group()，后续新增并行组都应经由该工厂；同时暴露了一类容易被忽略的模式——单 rank 组上的裸 collective 消费点。对团队：PR提供了完整的复现数据、精度测试和 CI 归因，降低了 NR 类显存问题的排查成本；NPU CI lane 的预存在故障也被顺带记录在案。 - 风险标记：backend 语义变更，隐藏消费点需排查，依赖 NCCL 版本行为

关联脉络

- PR #35626 [diffusion] fix: keep large vocab tables in host memory under layerwise offload: 同属 diffusion 模块的显存优化线：一个解决大 vocab 表驻留 host，一个解决并行组 NCCL 通信预留，共同压低单卡显存水位。
- PR #35618 [diffusion] UX: report where a component's weights are: 同为 diffusion 运行时资源治理与可观测性改进，涉及 loader 与 layerwise_offload 的权重驻留位置报告。
- PR #35612 [diffusion] fix: keep Cosmos3 T=1 fusion on Blackwell: 同为 diffusion 模块的定点修复 PR，反映 diffusion 子系统的近期维护节奏与回归验证方式。