PR #37070 完整报告

sgl-project/sglang

Scatter mm embeddings with row index_copy_ instead of masked_scatter_ to cut transient GPU memory

合并时间：2026-08-30 15:15

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/37070

执行摘要

- 一句话：多模态 embedding 改用 index_copy_ 合并，削减 prefill 瞬时显存
- 推荐动作：值得精读。这是一个典型的“PyTorch 高 Layer 算子隐藏瞬时内存分配”的优化案例，展示了如何用 cumsum + 哨兵槽位实现低内存且 fail-loud 的 scatter 语义。关注两点设计：一是刻意避免 torch.where/nonzero 以维持无同步路径，二是用 num_src_rows 毒化槽位把行数不匹配变成设备端断言而非静默损坏。若后续在 CUDA 上补齐同样的回归测试，风险将进一步收敛。

功能与动机

PR body 明确指出：embed_mm_inputs 中合并多模态 embedding 时使用 masked_scatter_，内部会物化展开后的 [num_tokens, hidden] bool mask 和 int64 prefix-sum，约 9 字节每 num_tokens x hidden 元素，在长上下文多模态 batch 下每个 prefill chunk 会出现多 GiB 的分配峰值，可能把紧凑部署推入 OOM。作者的核心诉求是在不改动 kernel 速度的前提下，削减该路径的峰值显存。

实现拆解

1. 定位变更入口：在 python/sglang/srt/managers/mm_utils.py 的 embed_mm_inputs 第 4 步“scatter embeddings into input embedding”中，原有的嵌套闭包 _scatter 使用 dest.masked_scatter_(mask.expand_as(dest), src.to(...))，是瞬时显存峰值的来源。
2. 新增模块级助手函数 _scatter_mm_embedding：将 [num_tokens, 1] 的 mask 展平后做 torch.cumsum 得到每个 True 位置的目标行号（rank），用 masked_fill 把 False 行折叠到哨兵槽 num_src_rows，再通过 rows.scatter_ 构建 src 行到 dest 行的映射表，最后用 dest.index_copy_ 完成合并。瞬态张量只有 rank 与 rows 两个 O(num_tokens) 的 long 张量，且全程无 torch.where/nonzero，不触发 host-device 同步。
3. 替换两处调用点：主路径 _scatter_mm_embedding(dest=input_embeds, ...) 与 deepstack 路径 _scatter_mm_embedding(dest=input_deepstack_embeds, ...) 统一走新 helper，原嵌套 _scatter 闭包被删除。
4. 错误语义兜底：mask 与 src 行数不匹配时，未匹配行会折叠到毒化槽位，在 scatter_ 或 index_copy_ 中触发设备端越界断言，而非静默写坏数据。
5. 测试配套：新增 test/registered/unit/managers/test_mm_embed_scatter.py，注册到 base-b-test-cpu suite，用参数化覆盖 interleaved/block/all-true/all-false 四种 mask 模

式、bf16/fp32 两种 src dtype、两种宽度，共 17 个用例，验证与 masked_scatter_ 的逐位相等，并额外验证行数不匹配时抛出 RuntimeError/IndexError。第二个 commit 为测试文件补充 __main__ pytest 入口，满足 CI registry 的 sanity check。

关键文件：

- python/sglang/srt/managers/mm_utils.py（模块 多模态；类别 source；类型 core-logic；符号 _scatter_mm_embedding, embed_mm_inputs, _scatter）：核心源码文件：新增模块级 _scatter_mm_embedding 替代 masked_scatter_，并把主 embedding 与 deepstack embedding 两条 scatter 路径统一到新实现，是本次显存优化的全部逻辑所在。
- test/registered/unit/managers/test_mm_embed_scatter.py（模块 多模态；类别 test；类型 test-coverage；符号 make_mask, test_scatter_matches_masked_scatter_bitwise, test_scatter_row_count_mismatch_fails_loud）：新增测试文件：通过 17 个参数化用例把新实现与 masked_scatter 语义做逐位对比，并覆盖行数不匹配的 fail-loud 行为，是本次性能优化不回退正确性的关键保障。

关键符号：_scatter_mm_embedding, embed_mm_inputs

关键源码片段

python/sglang/srt/managers/mm_utils.py

核心源码文件：新增模块级 _scatter_mm_embedding 替代 masked_scatter_，并把主 embedding 与 deepstack embedding 两条 scatter 路径统一到新实现，是本次显存优化的全部逻辑所在。

```
# 模块级 helper：用 row index_copy_ 取代 masked_scatter_ 完成多模态 embedding 合并。
# masked_scatter_ 需要展开 [num_tokens, hidden] 的 bool mask 并做 int64 prefix-sum，
# 瞬时显存约 9 B/ 元素；这里把瞬态压到 O(num_tokens) 且全程无 host-device 同步。
def _scatter_mm_embedding(
    dest: torch.Tensor, mask: torch.Tensor, src: torch.Tensor
) -> None:
    # mask: [num_tokens, 1] bool；src: [num_mm_tokens, width]，按序列顺序排列。
    src = src.to(dest.device, dest.dtype)
    num_src_rows = src.size(0)
    flat_mask = mask.view(-1)
    # 每个 True 位置获得其在所有 True 位置中的序号（0 起），即目标行号。
    ranks = torch.cumsum(flat_mask, dim=0) - 1
    # False 行折叠进哨兵槽 num_src_rows；若 mask 与 src 行数不匹配，
    # 会在后续 scatter_/index_copy_ 中触发设备端断言，而非静默写坏数据。
    ranks = ranks.masked_fill(~flat_mask, num_src_rows)
    # rows 是“src 第 i 行应写入 dest 哪一行”的映射表，毒化值 dest.size(0) 用于越界兜底。
    rows = torch.full(
        (num_src_rows + 1,), dest.size(0), dtype=torch.long, device=dest.device
    )
    # 把 mask 中每个 True 位置的行号写进对应 rank 槽位。
    rows.scatter_(0, ranks, torch.arange(flat_mask.numel(), device=dest.device))
    # 按行号把多模态 embedding 拷贝进 text embedding；索引越界时在设备端直接报错。
    dest.index_copy_(0, rows[:num_src_rows], src)
```

test/registered/unit/managers/test_mm_embed_scatter.py

新增测试文件：通过 17 个参数化用例把新实现与 masked_scatter_ 语义做逐位对比，并覆盖行数不匹配的 fail-loud 行为，是本次性能优化不回退正确性的关键保障。

```
# 与 masked_scatter_ 语义逐位对比的回归测试，覆盖多种 mask 形态与 dtype。
@pytest.mark.parametrize("width", [8, 24])
@pytest.mark.parametrize("src_dtype", [torch.bfloat16, torch.float32])
@pytest.mark.parametrize(
    "mask_pattern", ["interleaved", "blocks", "all_true", "all_false"]
)
def test_scatter_matches_masked_scatter_bitwise(width, src_dtype, mask_pattern):
    torch.manual_seed(0)
    mask = _make_mask(mask_pattern)
    dest = torch.randn(NUM_TOKENS, width).to(torch.bfloat16)
    src = torch.randn(int(mask.sum()), width, dtype=src_dtype)

    # 基线：保持原 masked_scatter_ 语义，确保多模态行不被写坏。
    expected = dest.clone()
    expected.masked_scatter_(mask.expand_as(expected), src.to(expected.dtype))

    # 新实现：基于 cumsum 行索引的 index_copy_ 合并，结果必须逐位一致。
    actual = dest.clone()
    _scatter_mm_embedding(dest=actual, mask=mask, src=src)
    assert torch.equal(actual, expected)
```

评论区精华

该 PR 没有任何 reviewer 评论，issue 评论仅为 CI 触发指令（/tag-and-rerun-ci、/rerun-failed-ci）和 @houseroad 的注意提及。设计取舍主要写在 PR body 中：作者明确“No kernel-speed change expected or claimed; the win is the removal of the expanded-mask transient allocations”，并在内部多模态 serving 环境通过内存快照验证了显存峰值下降。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. 依赖设备端越界断言：fail-loud 行为依赖 scatter_/index_copy_ 对越界索引报错，而测试仅在 CPU 上运行，CUDA 端的设备侧断言行为未直接覆盖，需留意不同后端的一致性。
  2. mask 设备一致性：新 helper 未显式搬运 mask，torch.cumsum(flat_mask) 在 mask 所在设备执行，而 rows 显式建在 dest.device，若将来某条路径传入 CPU mask 会直接报跨设备错误；原 masked_scatter_ 同样要求 mask 与 dest 同设备，因此未引入新的语义缺口，但新代码更早暴露问题。
  3. kernel 数量增加：原实现是单算子 masked_scatter_，新实现变为 cumsum + scatter_ + index_copy_ 三个算子，kernel 启动开销略增；对极短序列可能收益不明显，但对长上下文是显存收益远大于启动开销。

4. 核心 prefill 路径变更：embed_mm_inputs 是所有多模态请求 prefill 的必经路径，任何回归都会影响全部多模态模型；好在有逐位对比测试兜底，且语义严格对齐。 - 影响：影响范围集中在多模态长上下文 prefill 路径：每个 prefill chunk 的瞬时显存分配从 O(num_tokens x hidden) 降为 O(num_tokens)，可显著缓解大 batch 长上下文场景下的 OOM 风险；对短序列与纯文本请求无实质影响。行为上与 masked_scatter_ 逐位一致，因此对端到端输出无任何变化，用户透明。对团队而言，新增了一个可复用的 scatter 工具函数和一套高覆盖回归测试，也为后续其他需要规避 masked_scatter_/torch.where 瞬时内存的路径提供了参考范式。 - 风险标记：核心路径变更，缺少 GPU 端测试，依赖设备端越界断言

关联脉络

- PR #37116 [diffusion] perf: absorb Qwen-Image output projection biases: 同属多模态 / 扩散模型推理路径上的性能与显存优化，反映该方向持续演进，可一起追踪性能收益积累。
- PR #36991 [Diffusion] Add exact component precision overrides: 同在 multimodal 组件加载与 embedding 处理链路，涉及 embedding 精度与驻留行为，与本 PR 的多模态 embedding 合并路径有上下文关联。