PR #25299 完整报告

sgl-project/sglang

[NSA] Avoid repeated NSA MQA logits memory queries

合并时间：2026-05-20 07:04

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/25299

执行摘要

- 一句话：缓存 MQA logits 内存预算，避免重复 host sync
- 推荐动作：本 PR 展示了如何通过缓存避免 host-device 同步来优化延迟敏感路径，设计简洁效果明显。建议阅读以学习性能优化技巧，尤其对于涉及 GPU 内存查询的场景。同时，关注缓存预算的计算方式，可参考其双预算约束设计。

功能与动机

NSA indexer 在大的 MQA logits 路径上会频繁调用 torch.cuda.mem_get_info，这是一个 host-syncing 查询，位于延迟敏感的 prefill 路径中，影响性能。（Reported-by: @samuellees）

实现拆解

1. 在 Indexer 类中添加类级常量，将原先硬编码的 0.5、0.3、8_000_000 等提取为 _MQA_LOGITS_FREE_MEM_FRACTION、_MQA_LOGITS_TOTAL_MEM_FRACTION、_MQA_LOGITS_STATIC_SKIP_ELEMS。
2. 新增 _get_mqa_logits_budget_bytes(self, device_index) 方法：首先检查 _mqa_logits_budget_bytes 字典缓存；若未缓存，则根据设备总内存和 mem_fraction_static 计算静态服务内存预算，并在 CUDA graph capture 模式返回静态预算而不缓存；否则，执行一次 torch.cuda.mem_get_info 获取当前空闲内存，取两者较小值作为预算，并缓存到字典中。
3. 重构 _should_chunk_mqa_logits 方法：将参数 device 改为 device_index；使用缓存的预算替代每次调用 mem_get_info；将原始判断条件 logits_bytes * 2 > free_mem or logits_bytes > total_mem * 0.3 简化为 logits_bytes > logits_budget_bytes。
4. 在 _get_topk_ragged 中复用 cu_seqlens_q 的分配，避免在 chunked paged 路径的循环内部重复分配。

关键文件：

- python/sglang/srt/layers/attention/nsa/nsa_indexer.py（模块 NSA 模块；类别 source；类型 core-logic；符号 _get_mqa_logits_budget_bytes, _should_chunk_mqa_logits）：所有变更集中于此文件：类常量、缓存方法和 chunk 逻辑重构。

关键符号：_get_mqa_logits_budget_bytes, _should_chunk_mqa_logits

关键源码片段

python/sglang/srt/layers/attention/nsa/nsa_indexer.py

所有变更集中于此文件：类常量、缓存方法和 chunk 逻辑重构。

```
class Indexer(MultiPlatformOp):
    # 类常量，替代魔数
    _MQA_LOGITS_BYTES_PER_ELEM = 4
    _MQA_LOGITS_STATIC_SKIP_ELEMS = 8_000_000 # 跳过小 batch 的静态阈值
    _MQA_LOGITS_FREE_MEM_FRACTION = 0.5 # 空闲内存使用上限比例
    _MQA_LOGITS_TOTAL_MEM_FRACTION = 0.3 # 总内存使用上限比例
    _mqa_logits_budget_bytes: Dict[int, int] = {} # 每设备缓存字典

    def _get_mqa_logits_budget_bytes(self, device_index: int) -> int:
        """获取缓存的内存预算，避免每次 host sync"""
        # 优先返回缓存值
        cached_budget = self._mqa_logits_budget_bytes.get(device_index)
        if cached_budget is not None:
            return cached_budget

        total_mem = torch.cuda.get_device_properties(device_index).total_memory
        # 基于总内存的预算
        total_mem_budget = int(total_mem * self._MQA_LOGITS_TOTAL_MEM_FRACTION)
        mem_fraction_static = get_global_server_args().mem_fraction_static
        if mem_fraction_static is None:
            static_budget = total_mem_budget
        else:
            static_free_mem = int(total_mem * max(0.0, 1.0 - mem_fraction_static))
            # 静态预算取空闲内存比例与总内存比例的较小值，避免过度松弛
            static_budget = min(
                int(static_free_mem * self._MQA_LOGITS_FREE_MEM_FRACTION),
                total_mem_budget,
            )
        static_budget = max(1, static_budget)

        # CUDA graph capture 阶段不缓存，返回静态预算
        if get_is_capture_mode():
            return static_budget

        # 首次非 capture 时查询一次并缓存
        free_mem, _ = torch.cuda.mem_get_info(device_index)
        budget_bytes = min(
            int(free_mem * self._MQA_LOGITS_FREE_MEM_FRACTION), static_budget
        )
        budget_bytes = max(1, budget_bytes)
        self._mqa_logits_budget_bytes[device_index] = budget_bytes
        return budget_bytes

    def _should_chunk_mqa_logits(
```

```
        self, num_q: int, num_k: int, device_index: int
    ) -> Tuple[bool, int]:
        """判断是否需要分块计算 MQA logits 以避免 OOM
        Returns: (need_chunk, logits_budget_bytes)
        """
        # 小 batch 直接跳过，避免不必要的开销
        if num_q * num_k < self._MQA_LOGITS_STATIC_SKIP_ELEMS:
            return False, 0

        logits_bytes = num_q * num_k * self._MQA_LOGITS_BYTES_PER_ELEM
        logits_budget_bytes = self._get_mqa_logits_budget_bytes(device_index)
        need_chunk = logits_bytes > logits_budget_bytes
        return need_chunk, logits_budget_bytes
```

评论区精华

- samuellees要求将魔数 0.3 和 0.5 提取为类常量，作者已采纳并添加 _MQA_LOGITS_FREE_MEM_FRACTION 和 _MQA_LOGITS_TOTAL_MEM_FRACTION。
- Fridge003关注缓存预算在首次大 batch 时可能过松，导致后续 chunk 不足而 OOM。作者解释：缓存仅在非 CUDA graph capture 路径上由大窗口触发，且结果会被工作量无关的静态服务内存头寸（mem_fraction_static）限制，不会过松。该问题已解决。
- 将魔数提取为类常量 (style): 作者添加了对应常量并回复 'Adjusted'。
- 缓存预算的过松风险 (design): 作者解释缓存仅在非 capture 路径上计算，且受 static budget cap，不会过松。

风险与影响

- 风险：
  1. 缓存过时风险：缓存的值可能在长期运行中因内存碎片或并发请求而变得不准确，但通过 mem_fraction_static 和总内存比例双重限制，OOM 风险较低。
  2. 首次查询仍有一次 sync：第一个大 batch 非 capture 路径仍有一次 mem_get_info 开销，但仅一次，可接受。
  3. 测试覆盖不足：未添加直接单元测试，但已有 E2E 测试覆盖 NSA 路径，且性能测试验证了正确性。
  4. 多线程安全：_mqa_logits_budget_bytes 为类字典，device_index 唯一且写后读，无竞争条件。
- 影响：
  - 用户：DeepSeek 模型在长上下文预填充阶段获得 6-10% 延迟改善和吞吐提升，尤其大 batch 场景受益显著。
  - 系统：减少 GPU 同步次数，提高调度效率和 GPU 利用率。
  - 团队：代码更易维护，魔数提取为常量便于未来调参。
  - 风险标记：缓存依赖 GPU 属性，缺少测试覆盖，核心路径变更

关联脉络

- 暂无明显关联 PR