PR #28076 完整报告

sgl-project/sglang

[perf] remove several h2d sync

合并时间：2026-06-13 11:30

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/28076

执行摘要

- 一句话：移除 CUDA 同步点，优化 tensor 创建延迟
- 推荐动作：值得合并，这是一个清晰、低风险的优化，遵循了 PyTorch 官方推荐的做法。核心设计决策——在热路径上使用 pin memory 和异步传输——可以推广到其他张量创建上。

功能与动机

PR 描述中提供了 CUDA 分析的截图（Before/After），显示 from_schedule_batch 中存在多个 cudaStreamSynchronize 调用，导致高延迟。通过避免在 torch.tensor(..., device=device) 中创建 CUDA 张量时隐含的同步，可以显著减少开销。

实现拆解

1. 在 SamplingBatchInfo.from_schedule_batch 中应用 pin_memory + non_blocking：新增 from sglang.srt.utils.common import is_pin_memory_available 导入，调用 _pin = is_pin_memory_available(device) 检查 pin memory 可用性。随后将所有 torch.tensor(..., device=device) 替换为两步模式：先在 CPU 上创建 pin_memory=True 的张量，再对结果调用 .to(device, non_blocking=True)，从而将 CPU->GPU 传输异步化。涉及 temperatures、top_ps、top_ks、min_ps、sampling_seed 五个张量。
2. 在 write_cache_indices 中应用相同模式：在 python/sglang/srt/mem_cache/common.py 中，将 prefix_pointers = torch.tensor(..., device=..., dtype=torch.uint64) 拆分为 CPU pin_memory 分配后，再通过 .to(device, non_blocking=True) 传给 Triton 内核。

关键文件：

- python/sglang/srt/sampling/sampling_batch_info.py（模块 采样器；类别 source；类型 dependency-wiring）：核心改动：将 from_schedule_batch 中 5 个张量的创建从同步改为异步，移除了大量 cudaStreamSynchronize 调用。
- python/sglang/srt/mem_cache/common.py（模块 缓存层；类别 source；类型 dependency-wiring）：次要改动：在 write_cache_indices 中对 prefix_pointers 张量应用相同的 pin_memory + non_blocking 模式。

关键符号：SamplingBatchInfo.from_schedule_batch, write_cache_indices

关键源码片段

python/sglang/srt/sampling/sampling_batch_info.py

核心改动：将 from_schedule_batch 中 5 个张量的创建从同步改为异步，移除了大量 cudaStreamSynchronize 调用。

```
# python/sglang/srt/sampling/sampling_batch_info.py
# 关键方法：SamplingBatchInfo.from_schedule_batch
@classmethod
def from_schedule_batch(cls, batch: ScheduleBatch, vocab_size: int):
    global_server_args = get_global_server_args()
    enable_deterministic = global_server_args.enable_deterministic_inference

    reqs = batch.reqs
    device = batch.device
    _pin = is_pin_memory_available(device) # 检查当前设备是否支持 pin memory

    temperatures = (
        torch.tensor(
            [r.sampling_params.temperature for r in reqs],
            dtype=torch.float,
            pin_memory=_pin, # 先分配 pin memory 的 CPU 张量
        )
        .to(device, non_blocking=True) # 异步传输：消除 host-to-device 同步
        .view(-1, 1)
    )
    top_ps = torch.tensor(
        [r.sampling_params.top_p for r in reqs],
        dtype=torch.float,
        pin_memory=_pin,
    ).to(device, non_blocking=True)
    top_ks = torch.tensor(
        [r.sampling_params.top_k for r in reqs],
        dtype=torch.int32,
        pin_memory=_pin,
    ).to(device, non_blocking=True)
    min_ps = torch.tensor(
        [r.sampling_params.min_p for r in reqs],
        dtype=torch.float,
        pin_memory=_pin,
    ).to(device, non_blocking=True)
    sampling_seed = (
        torch.tensor(
            [
                (
                    r.sampling_params.sampling_seed
                    if r.sampling_params.sampling_seed is not None
                    else 42
                )
                for r in reqs
```

```
            ],
            dtype=torch.int64,
            pin_memory=_pin,
        ).to(device, non_blocking=True)
        if enable_deterministic
        else None
    )
```

python/sglang/srt/mem_cache/common.py

次要改动：在 write_cache_indices 中对 prefix_pointers 张量应用相同的 pin_memory + non_blocking 模式。

```
# python/sglang/srt/mem_cache/common.py
# 函数：write_cache_indices
if support_triton(get_global_server_args().attention_backend):
    prefix_pointers = torch.tensor(
        [t.data_ptr() for t in prefix_tensors],
        dtype=torch.uint64,
        pin_memory=is_pin_memory_available(req_to_token_pool.device), # CPU pin memory
    ).to(req_to_token_pool.device, non_blocking=True) # 异步传输至 GPU
    write_req_to_token_pool_triton[(req_pool_indices_tensor.shape[0],)](
        req_to_token_pool.req_to_token,
        req_pool_indices_tensor,
        prefix_pointers,
        prefix_lens_tensor,
        seq_lens_tensor,
        extend_lens_tensor,
        out_cache_loc,
        req_to_token_pool.req_to_token.shape[1],
    )
```

评论区精华

该 PR 没有 review 评论或讨论线程。

- 暂无高价值评论线程

风险与影响

- 风险：低风险。更改仅限于两个辅助类，仅在推理路径中优化张量创建方式，不改变模型权重、业务逻辑或外部 API。is_pin_memory_available 会兼容不支持 pin memory 的设备，且 non_blocking 传输是标准 PyTorch 模式。回归风险极低。
- 影响：对用户的正面性能提升，尤其是在小批量或低延迟场景下。对系统的改动集中在内核初始化步骤中，不会影响数据流或正确性。团队维护成本低，因为改动遵循了标准 PyTorch 最佳实践。
- 风险标记：缺少测试覆盖

关联脉络

- 暂无明显关联 PR