PR #34124 完整报告

sgl-project/sglang

[diffusion] perf_logger: SYNC_STAGE_PROFILING must drain the GPU queue for stage records too (fixes 2-3x inflated DecodingStage readings)

合并时间：2026-08-09 09:49

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/34124

执行摘要

- 一句话：修复 diffusion 阶段计时未同步 GPU 队列导致的 2-3 倍虚高
- 推荐动作：值得关注 diffusion 性能分析和 GPU 异步计时工具的同学精读：修复虽小，但 _maybe_sync_device() 在计时起点与终点双向同步的语义清晰解决了异步队列归属问题，新增测试也精巧复现了队列尾部泄漏场景。对普通 SGLANG 用户无需跟进。可留意此修复落地后，依赖 stage 计时做消融对比的 diffusion 优化 PR（如 #34126、#34015）读数会更容易解释。

功能与动机

PR body 明确指出 SGLANG_DIFFUSION_SYNC_STAGE_PROFILING=1 承诺同步 stage 计时，但 StageProfiler 的入口和出口 synchronize() 都被 _should_record_as_step() 门控，只有 denoising_step_* 记录会同步；DenoisingStage 在最后一步上下文关闭后仍会继续提交 scheduler、guidance、latent 后处理等工作，其设备队列尾部被稍后第一个阻塞的 stage（典型是 DecodingStage）吸收，导致 DecodingStage 墙钟读数包含上一阶段尾部，在三模型对比中 stage 读数高出 2-3 倍，并引发了 prH 中所谓“95 ms mystery”的幽灵排查。由于该 flag 默认关闭，此问题只影响显式开启同步 profiling 的诊断场景。

实现拆解

1. 定位根因：python/sglang/multimodal_gen/runtime/utils/perf_logger.py 中 StageProfiler.__enter__ 与 __exit__ 原本把 torch.get_device_module().synchronize() 包在 SGLANG_DIFFUSION_SYNC_STAGE_PROFILING=1 && self._should_record_as_step() 条件内，只有 step 记录同步。
2. 新增同步 helper：新增 _maybe_sync_device() 方法，将环境变量检查与设备可用判断收拢到一处，并去掉 _should_record_as_step() 门控；__enter__ 计时起点与 __exit__ 计时终点都改用它。
3. 语义调整：入口同步先于 start_time，把上游 stage 排队但尚未执行的工作 drain 掉，使其归属到上游；出口同步先于 execution_time_s 计算，把当前 stage 自身排队的尾部纳入自身。两步合起来消除“后一个 stage 替前一个 stage 买单”的窗口。
4. 测试配套：新增 test/registered/kernels/ops/diffusion/test_stage_profiler_sync.py，用 torch.cuda._sleep 校准约 0.5 s 排队工作，producer stage 只排队不等待、consumer stage 做阻塞小 op；断言 producer > 250 ms、consumer < 100 ms。该测试在 main 上

失败（producer 0.0115 ms、consumer 471.6 ms），修复后通过，并以 register_cuda_ci(est_time=30, stage='base-b-kernel-unit', runner_config='1-gpu-large') 注册进 CI。

5. 实测验证：H200 + Krea-2-Turbo 1024×1024 / 8 steps 下，DecodingStage 从 0.164-0.166 s 降至 0.067-0.068 s，DenoisingStage 从 1.314-1.315 s 升至 1.461-1.462 s，总时长基本不变，证明只调整归属、不改变行为。

关键文件：

- python/sglang/multimodal_gen/runtime/utils/perf_logger.py（模块 性能日志；类别 source；类型 core-logic；符号 StageProfiler, _maybe_sync_device, _should_record_as_step）：核心修复点：StageProfiler 的同步逻辑从 step 专用扩展为所有记录统一同步，新增 _maybe_sync_device 并在计时起点 / 终点调用。
- test/registered/kernels/ops/diffusion/test_stage_profiler_sync.py（模块 回归测试；类别 test；类型 test-coverage；符号 test_stage_entry_sync_excludes_previous_stage_tail）：新增回归测试，用 producer/consumer 模式捕获上一 stage 队列尾部泄漏到下一 stage 的问题，main 上红、修复后绿。

关键符号：_maybe_sync_device, test_stage_entry_sync_excludes_previous_stage_tail

关键源码片段

python/sglang/multimodal_gen/runtime/utils/perf_logger.py

核心修复点：StageProfiler 的同步逻辑从 step 专用扩展为所有记录统一同步，新增 _maybe_sync_device 并在计时起点 / 终点调用。

```
# python/sglang/multimodal_gen/runtime/utils/perf_logger.py
# 同步逻辑修复：新增 _maybe_sync_device()，并在每个记录的计时起点和终点调用
class StageProfiler:
    def _should_record_as_step(self) -> bool:
        # 只决定记录进 step 还是 stage，不再参与同步门控
        return self.record_as_step or self.stage_name.startswith('denoising_step_')

    def _maybe_sync_device(self):
        # 开启 SGLANG_DIFFUSION_SYNC_STAGE_PROFILING=1 时强制 drain GPU 队列。
        # 历史上只有 step 记录同步，stage 记录不同步，导致上游 stage 的
        # 排队尾部被后续第一个阻塞的 stage 吸收，读数虚高 2-3 倍。
        if (
            os.environ.get('SGLANG_DIFFUSION_SYNC_STAGE_PROFILING', '0') == '1'
            and torch.get_device_module().is_available()
        ):
            torch.get_device_module().synchronize()

    def __enter__(self):
        # ... start 日志省略 ...
        if (self.log_timing and self.metrics) or self.log_stage_start_end:
            self._maybe_sync_device() # 计时前先 drain，把上游排队工作归给上游
            self.start_time = time.perf_counter()
```

```
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        if not ((self.log_timing and self.metrics) or self.log_stage_start_end):
            return False
        self._maybe_sync_device() # 计时结束前 drain，纳入当前 stage 自身排队尾部
        execution_time_s = time.perf_counter() - self.start_time
        # ... 异常日志、memory snapshot、record_step / record_stage 省略 ...
        return False
```

test/registered/kernels/ops/diffusion/test_stage_profiler_sync.py

新增回归测试，用 producer/consumer 模式捕获上一 stage 队列尾部泄漏到下一 stage 的问题，main 上红、修复后绿。

```
# test/registered/kernels/ops/diffusion/test_stage_profiler_sync.py
# 复现队列尾部泄漏：producer 排队约 0.5 s 不等待，consumer 做阻塞小 op
def test_stage_entry_sync_excludes_previous_stage_tail(monkeypatch):
    monkeypatch.setenv('SGLANG_DIFFUSION_SYNC_STAGE_PROFILING', '1')
    logger = init_logger(__name__)
    metrics = RequestMetrics('stage-sync-test')

    # 校准 torch.cuda._sleep 的真实时长，目标约 0.5 s
    torch.cuda.synchronize()
    t0 = time.perf_counter()
    torch.cuda._sleep(10_000_000)
    torch.cuda.synchronize()
    cycles = int(10_000_000 / max(time.perf_counter() - t0, 1e-9) * 0.5)

    # producer 只排队不等待，模拟 DenoisingStage 的队尾工作
    with StageProfiler('producer', logger, metrics, perf_dump_path_provided=True):
        torch.cuda._sleep(cycles)
    # consumer 的第一个阻塞 op 在修复前会吸收 producer 的尾部
    with StageProfiler('consumer', logger, metrics, perf_dump_path_provided=True):
        torch.ones(8, device='cuda').sum().cpu()

    producer_ms = metrics.stages['producer']
    consumer_ms = metrics.stages['consumer']
    assert producer_ms > 250, f'queued work not attributed to producer: {metrics.stages}'
    assert consumer_ms < 100, f'producer tail leaked into consumer: {metrics.stages}'
```

评论区精华

该 PR 没有 code review 评论，技术讨论集中在 PR body 与作者的 CI 说明中。作者 BBuf 强调这是测量基础设施的正确性修复：flag 默认关闭、生产路径不受影响；并说明剩余红色 CI lane 均为仓库已知的非必需 lane——multimodal-gen-test-2-gpu-amd / -2-npu-a3 的 total_partitions (3) must be >= standalone files (7) 为预存 harness 配置错误，pr-test-{amd,extra,npu,}-finish 聚合器同样在近期合并的 #34008、#34085 上为红，与本次 diff 无关。

- CI 状态与预存失败 lane 说明 (other): CI 全绿，遗留红色与本次 diff 无关，不阻塞合并。

风险与影响

- 风险：
  1. 功能默认关闭（flag=0 时 _maybe_sync_device() 直接返回），生产推理路径不受影响；但一旦开启 SGLANG_DIFFUSION_SYNC_STAGE_PROFILING=1，每个 stage/step 记录前后都会插入 torch.get_device_module().synchronize()，会阻断 GPU 异步流水，可能使同步点附近的计时略有失真——这是诊断的固有代价。
  2. 新测试仅覆盖 CUDA（skipif not cuda）；_maybe_sync_device() 在 NPU、AMD 等后端的行为未验证，虽然环境判断统一经由 torch.get_device_module()。
  3. 测试基于 torch.cuda._sleep 校准并采用宽松阈值（producer > 250 ms、consumer < 100 ms），在慢速或高负载 CI 节点上仍可能有抖动，但 main 上红绿对比明确。
  4. 修复不改变任何生成阶段行为，不会影响图像输出质量。 - 影响：用户侧：仅显式开启 SGLANG_DIFFUSION_SYNC_STAGE_PROFILING=1 的 diffusion 性能分析用户会看到变化——stage 计时归属变准确，前序 stage 的队尾工作不再被计入后续 stage。系统侧：生产路径字节级不变，诊断路径新增 opt-in 同步开销，可忽略。团队侧：结束 prH “95 ms” 类幽灵耗时排查，使后续 diffusion 性能优化（如 #34126、#34015）的 benchmark 前后对比数据更可信；新增测试也为同类计时基础设施提供了回归保护。综合影响为低 - 中，主要惠及性能分析场景。 - 风险标记：opt-in 诊断路径变更，同步开销影响异步测量，仅 CUDA 测试覆盖

关联脉络

- PR #34085 [diffusion] Clean up kernels and shared fast paths: 与本次修复处于同一 diffusion 诊断 / 阶段计时链路；PR 评论中将其列为 CI 红色 lane 的先例，且该 PR 的 denoising 重构直接影响阶段计时归属。
- PR #34126 [diffusion] FLUX.1: route the adaLN LN+modulate sites through the bit-exact fused LayerNorm+modulate kernel (H200 1024^2 lossless denoise -1.2%, e2e wall -2.9%): FLUX.1 性能优化依赖准确的 DenoisingStage/DecodingStage 计时来量化收益；本修复修正 stage 归属后，此类优化前后对比更可信。
- PR #34015 [diffusion] Sana: bit-exact fused aten LayerNorm+modulate under BCG (H200 denoise -4.8%): Sana 融合 adaLN 内核的加速同样基于 stage 计时数据，与本次修复共享同一测量基础。