PR #24370 完整报告

sgl-project/sglang

Profiling Enhancements [1/3]: cuda graph profile traces

合并时间：2026-08-06 18:19

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/24370

执行摘要

- 一句话：CUDA 图捕获阶段新增按 batch size 的 profile trace
- 推荐动作：值得精读，特别是可观测性基建的设计：把 torch profiler 的 schedule 与 CUDA graph 捕获循环的步进对齐（wait=2/warmup=0/active=1）这一思路可以复用到任何「分阶段捕获 + 分片导出」的场景；双环境变量共存时「新特性让位于旧行为」的优先级设计、输出目录单一事实来源、以及 getattr 防御式接入 backend 的做法，都是兼顾兼容性与可扩展性的好示范。建议阅读时重点关注 decode_cuda_graph_runner.py 的 _init_profile_context_and_memory_record() 与 full_cuda_graph_backend.py 的 capture_one() 之间的契约（_profiler 属性）。

功能与动机

PR body 明确说明：启动时 runner 为每个 batch size 捕获一张 CUDA graph，服务时按当前 batch size 回放对应图，因此捕获阶段看到的 shape 与 kernel 恰好就是生产环境实际执行的路径。按 batch size 分别记录 profiler trace，可以离线检查每个形状的 kernel（如 aiter fused_moe、attention）而不必对线上运行做 profiling，为捕获阶段这一不透明启动过程提供 per-shape 可见性。

实现拆解

1. 环境变量与开关：python/sglang/srt/environ.py 新增 SGLANG_GRAPH_BATCH_CAPTURE（EnvBool(False)，默认关闭）；decode_cuda_graph_runner.py 新增 _graph_batch_capture_active()，其语义为「per-bs 模式激活当且仅当新变量开启且原变量 SGLANG_ENABLE_CUDA_GRAPH_CAPTURE_TRACE 未开启」，确立双模式优先级：原单 trace 模式优先。
2. 解码图 runner 改造：_init_profile_context_and_memory_record() 分叉为两条路径——per-bs 模式构建带 schedule(wait=2, warmup=0, active=1, repeat=0) 的 profiler，开启 record_shapes、with_stack、with_flops、profile_memory，并绑定 on_trace_ready 回调（按 {runner_name}_bs_{bs}_rank{rank}.json.gz 命名，_profile_bs_list 预反转以匹配从大到小的捕获顺序，_profile_bs_idx 计数）；原模式维持无 schedule 的单个 profiler。capture() 中初始化 self._profiler = None，仅在 per-bs 模式将其置为 profiler context，结束后复位，避免对其他路径产生副作用。
3. backend 步进：full_cuda_graph_backend.py 在 __init__ 中保存 self._cuda_graph_runner 引用；capture_one() 用双重 getattr 防御式读取 _profiler——

只有当 enable_profile_cuda_graph 为真且 runner 存在 _profiler 属性时才执行 profiler.step()：两次 warmup 各 step 一次（对应 wait=2 跳过 dummy 运行），真实 capture 后再 step 一次触发 on_trace_ready 落盘。非 profiling 路径调用序列与旧版完全一致（2 次 warmup + 1 次 capture，无任何 step）。

4. 输出目录统一：profile_utils.py 新增 GRAPH_CAPTURE_PROFILE_DIRNAME 常量与 graph_capture_profile_dir() 作为唯一输出目录入口，原 export_cuda_graph_capture_trace() 与 per-bs 模式共用 <SGLANG_TORCH_PROFILER_DIR>/graph_capture_profile/；文件名按 runner 类名与 TP rank 命名，确保 EAGLE3 的 target/draft/draft-extend 及多 rank 互不覆盖。
5. 测试与文档：新增 test_decode_cuda_graph_runner.py（6 用例，覆盖目录创建、反转 bs 列表与索引初始化、profiler 构造参数、默认目录回退、原模式与优先级、on_trace_ready 命名与 rank 后缀）和 test_full_cuda_graph_backend.py（5 用例，覆盖无 profiling 时 2 warmup + 1 capture 无 step、getattr 缺失时优雅降级、profiler 步进次数、capture 不再包 record_function），均为 CPU-only 且注册到 base-a-test-cpu；docs/docs/developer_guide/benchmark_and_profiling.mdx 补充两种环境变量的用法、输出位置与优先级说明。

关键文件：

- python/sglang/srt/model_executor/runner/decode_cuda_graph_runner.py（模块 图捕获器；类别 source；类型 core-logic；符号 _graph_batch_capture_active, _init_profile_context_and_memory_record, on_trace_ready, capture）：核心实现所在：新增 _graph_batch_capture_active() 双模式门控，_init_profile_context_and_memory_record() 分叉构建 scheduled profiler 与 on_trace_ready 回调，capture() 暴露并复位 _profiler 状态。该文件决定了 per-bs trace 功能的开关语义与命名契约。
- python/sglang/srt/model_executor/runner_backend/full_cuda_graph_backend.py（模块 图后端；类别 source；类型 core-logic；符号 capture_one, init）：per-bs 模式的执行端：capture_one() 在两次 warmup 后各 profiler.step() 一次、真实 capture 后再 step 一次，是 schedule 与捕获循环对齐的关键；同时用 getattr 保证非 profiling 路径零变化。
- python/sglang/srt/utils/profile_utils.py（模块 剖析工具；类别 source；类型 refactor；符号 graph_capture_profile_dir, export_cuda_graph_capture_trace）：提供单一事实来源的输出目录 graph_capture_profile_dir()，原单 trace 导出函数与 per-bs 模式共用路径，避免目录分裂；是 review 讨论后重构的落点。
- python/sglang/srt/environ.py（模块 环境变量；类别 source；类型 configuration；符号 SGLANG_GRAPH_BATCH_CAPTURE）：新增 SGLANG_GRAPH_BATCH_CAPTURE 环境变量声明，并注明与原 SGLANG_ENABLE_CUDA_GRAPH_CAPTURE_TRACE 的优先级关系，是功能开关的源头。
- test/registered/unit/model_executor/runner/test_decode_cuda_graph_runner.py（模块 图捕获器；类别 test；类型 test-coverage；符号 _make_fake_self, TestInitProfileBatchMode, _invoke, test_creates_graph_capture_profile_dir）：新增 6 个 CPU 单测，覆盖 per-bs 模式的目录创建、反转 bs 列表与索引初始化、profiler 参数与 schedule、默认目录回退、原模式与优先级、on_trace_ready 命名与 rank 后缀，是功能

正确性的主要保障。

- test/registered/unit/model_executor/runner_backend/test_full_cuda_graph_backend.py（模块 图后端；类别 test；类型 test-coverage；符号 _FakeGraphCtx, _make_backend, _make_runner, TestCaptureOneNoProfiling）：新增 5 个 CPU 单测，覆盖 capture_one 非 profiling 路径行为不变、getattr 缺失时优雅降级、profiler 步进次数与 record_function 移除，保证对核心捕获路径的侵入最小。
- docs/docs/developer_guide/benchmark_and_profiling.mdx（模块 开发文档；类别 docs；类型 documentation）：补充 CUDA graph capture 阶段 profiling 的用法文档：两个环境变量的差异、输出文件命名、前置条件与查看方式，是功能落地给用户的入口。

关键符号：_graph_batch_capture_active, _init_profile_context_and_memory_record, on_trace_ready, capture, capture_one, graph_capture_profile_dir, export_cuda_graph_capture_trace

关键源码片段

python/sglang/srt/model_executor/runner/decode_cuda_graph_runner.py

核心实现所在：新增 _graph_batch_capture_active() 双模式门控，_init_profile_context_and_memory_record() 分叉构建 scheduled profiler 与 on_trace_ready 回调，capture() 暴露并复位 _profiler 状态。该文件决定了 per-bs trace 功能的开关语义与命名契约。

```
# 双模式门控：per-bs trace 仅在 SGLANG_GRAPH_BATCH_CAPTURE 开启、且原单 trace 变量
# 未设置时生效（旧行为优先，保证向后兼容）。
def _graph_batch_capture_active(self) -> bool:
    return (
        envs.SGLANG_GRAPH_BATCH_CAPTURE.get()
        and not envs.SGLANG_ENABLE_CUDA_GRAPH_CAPTURE_TRACE.get()
    )


def _init_profile_context_and_memory_record(self):
    if self._graph_batch_capture_active():
        # per-bs 模式：为每个 batch size 构建一个带 schedule 的 profiler，
        # 由 FullCudaGraphBackend.capture_one 逐步 step，on_trace_ready 负责落盘。
        rank = get_parallel().tp_rank
        runner_name = type(self).__name__
        trace_dir = graph_capture_profile_dir()
        os.makedirs(trace_dir, exist_ok=True)

        # 捕获顺序为大 bs -> 小 bs，这里预先反转，保证回调命名与顺序一一对应。
        self._profile_bs_list = list(reversed(self.capture_bs))
        self._profile_bs_idx = 0

        def on_trace_ready(prof):
            bs = self._profile_bs_list[self._profile_bs_idx]
            trace_file = os.path.join(
```

```
                trace_dir, f"{runner_name}_bs_{bs}_rank{rank}.json.gz"
            )
            prof.export_chrome_trace(trace_file)
            logger.info(f"Saved trace for bs={bs} to {trace_file}")
            self._profile_bs_idx += 1

        profile_context = profile(
            activities=[ProfilerActivity.CPU, ProfilerActivity.CUDA],
            # wait=2 跳过两次 warmup 运行，active=1 只记录真实 capture 本身；
            # repeat=0 让每个 batch size 都独立触发一次 on_trace_ready。
            schedule=torch.profiler.schedule(wait=2, warmup=0, active=1, repeat=0),
            record_shapes=True,
            with_stack=True,
            with_flops=True,
            profile_memory=True,
            on_trace_ready=on_trace_ready,
        )
    else:
        # 原模式：单个无 schedule profiler 记录整个 capture 阶段，
        # 汇总表与内存快照在 _post_process_after_profile 中统一输出。
        profile_context = profile(
            activities=[ProfilerActivity.CPU, ProfilerActivity.CUDA],
            record_shapes=True,
        )
    torch.cuda.memory._record_memory_history()
    return profile_context
```

python/sglang/srt/model_executor/runner_backend/full_cuda_graph_backend.py

per-bs 模式的执行端：capture_one() 在两次 warmup 后各 profiler.step() 一次、真实 capture 后再 step 一次，是 schedule 与捕获循环对齐的关键；同时用 getattr 保证非 profiling 路径零变化。

```
def capture_one(self, shape_key, forward_fn, capture_inputs=None, post_warmup_hook=None):
    # 仅 per-bs 模式（--enable-profile-cuda-graph + SGLANG_GRAPH_BATCH_CAPTURE）下
    # runner 才会暴露 _profiler；双重 getattr 保证其余路径与旧行为完全一致。
    runner = self._cuda_graph_runner
    profiler = (
        getattr(runner, "_profiler", None)
        if getattr(runner, "enable_profile_cuda_graph", False)
        else None
    )

    # 两次 warmup：完成 kernel 加载与一次性初始化。
    # profiler.step() 让 schedule 跳过这两次（对应 wait=2），只记录真实 capture。
    for _ in range(2):
        self._device_module.synchronize()
        self._tp_group.barrier()
        forward_fn()
```

```
            if profiler is not None:
                profiler.step()
            if post_warmup_hook is not None:
                post_warmup_hook()

        graph = torch.cuda.CUDAGraph()

        graph_ctx: Callable[..., AbstractContextManager]
        if (
            self._memory_saver_adapter is not None
            and self._memory_saver_adapter.enabled
        ):
            graph_ctx = partial(
                self._memory_saver_adapter.cuda_graph,
                tag=GPU_MEMORY_TYPE_CUDA_GRAPH,
            )
        else:
            graph_ctx = self._device_module.graph

        with graph_ctx(cuda_graph=graph, pool=self._pool, stream=self._capture_stream):
            out = forward_fn()

        # capture 结束后 step 一次，触发 on_trace_ready 写出当前 bs 的 trace，
        # 对应 schedule 的 active=1 窗口。
        if profiler is not None:
            profiler.step()

        self._graphs[shape_key] = graph
        self._outputs[shape_key] = out
```

评论区精华

核心 review 交锋（评审人 HaiShaw）：

- profile_utils.py 上追问 "why don't you reuse or extend this function?"（export_cuda_graph_capture_trace）——作者经讨论后恢复原函数，不再替换原单 trace 能力，改为新增统一目录 helper graph_capture_profile_dir()，并引入独立 env 变量 SGLANG_GRAPH_BATCH_CAPTURE 让两模式并存且原模式优先。
- 对 per-bs 模式下 with_stack/with_flops/profile_memory 等开关是否应可选提出疑问，作者回应：这些细节多数用户最终都会用，默认开启，且只影响启动期 capture 阶段，不影响执行期 profiler。
- full_cuda_graph_backend.py 注释中环境变量名笔误（写成旧变量名），被 HaiShaw 指出后修正为 SGLANG_GRAPH_BATCH_CAPTURE。
- HaiShaw 明确要求补测试并附结果，作者随后新增两个测试文件 11 个用例并在 MI300X 容器中 e2e 验证。
- amd-bot 早期 CI 报告指出未加门控的版本在 NVIDIA/AMD/NPU 全后端 capture 崩溃（DecodeInputBuffers 缺 out_cache_loc_swa），最终版本通过

_graph_batch_capture_active() 及相关 gating 规避；amd-bot 同时警告新功能默认关闭、PR CI 未覆盖，作者以手动 e2e 验证回应。

- 是否复用 export_cuda_graph_capture_trace (design): 作者恢复原 export_cuda_graph_capture_trace，抽出 graph_capture_profile_dir() 统一目录，新增独立 env 变量 SGLANG_GRAPH_BATCH_CAPTURE 使两模式并存、原模式优先，既保留旧能力又新增 per-bs 能力。
- profiler 附加开关是否应可选 (design): 作者回应这些细节（stack、FLOPs、内存）大多数用户最终都会从 trace 中用到，默认开启；且该 profiler 只在 capture 阶段运行，执行期 profiler 仍由独立开关控制。评审未再异议。
- 注释中环境变量名笔误 (correctness): 作者确认 "Thanks for the catch, fixed" 并修正注释。
- 要求补充测试覆盖 (testing): 作者新增两个 CPU 单测文件共 11 个用例，覆盖非 profiling 路径行为不变、per-bs 步进、getattr 降级、优先级与命名；并在 MI300X 容器内跑通全部单测及端到端验证。
- amd-bot CI 报告：捕获崩溃与新功能未被 CI 覆盖 (correctness): 后续版本通过 _graph_batch_capture_active() 门控将新代码限定在 opt-in 路径，规避了崩溃；作者补充端到端验证说明 trace 文件正常产出。但 per-bs 真实 GPU 路径仍未被 CI 自动覆盖，合入前需人工冒烟。

风险与影响

- 风险：
  1. 早期版本存在启动崩溃：amd-bot 曾报告未门控代码在全部 GPU 后端 graph-capture 时因 out_cache_loc_swa 属性缺失而退出（exit -9）。最终版本已通过 _graph_batch_capture_active() gating 与等价行为测试规避，但 scheduled profiler + step() 的真实 GPU 路径仍未被 PR CI 覆盖，建议合入后在 NVIDIA/AMD 各做一次 --enable-profile-cuda-graph + SGLANG_GRAPH_BATCH_CAPTURE=1 启动冒烟。
  2. 启动期耗时增加：with_stack/with_flops/profile_memory 全开使 capture 阶段从约 37 s 增至约 47 s（MI355X 数据），虽仅 opt-in 且不影响 serving，但对启动时长敏感的场景需注意。
  3. 状态计数耦合：_profile_bs_idx 依赖 schedule 与捕获循环严格对齐，若某次 capture 异常导致 on_trace_ready 触发次数与 bs 数量不一致，可能产生文件命名错位或 IndexError；现单测仅覆盖正常路径。
  4. backend 覆盖范围：profiler 步进只接入 FullCudaGraphBackend；breakable 等后端未接入，per-bs 模式在这些后端下静默退化为无 per-bs trace（原汇总表仍输出），需在文档中注明。 - 影响：行为兼容性：默认路径与开启 --enable-profile-cuda-graph（不设 per-bs 变量）时行为与合入前一致，单测明确断言 2 warmup + 1 capture 且不触碰 profiler。用户影响：性能工程师现在可在启动期按 batch size 导出 Chrome trace，离线审查每个形状下的 kernel 与 shape，输出统一收敛到 <SGLANG_TORCH_PROFILER_DIR>/graph_capture_profile/。团队影响：这是 'Profiling Enhancements' 系列（1/3）的首块基石，为后续分阶段 profiling 能力奠定可观测性基建；6 组性能对照实验证明 serving 吞吐（1185.85–1193.75 tok/s）与 TPOT（25.94–26.18 ms）均无实质变化。 - 风险标记：opt-in 功能未被 CI 覆盖，启动期耗

时增加（with_stack 全开），状态计数耦合（_profile_bs_idx），仅 FullCudaGraphBackend 接入步进

关联脉络

- PR #33832 [CI] Remove profiling from nightly tests: 同为 profiling/ 可观测性基建方向：该 PR 移除 nightly 测试的 profiler 采集，而本 PR 为 CUDA graph 捕获阶段新增 profiling 能力，两者在 profiling 基础设施的使用与维护上互补。
- PR #33847 [CI] Restore the full prefill CUDA graph capture range in test launches: 涉及 CUDA graph capture 范围的 CI 配置调整，与本 PR 对 capture 阶段的可观测性增强属于同一 CUDA graph 捕获生命周期主题。
- PR #33776 [CI] Bound the CUDA graph capture range in test launches and lift the spec fixture's admission cap: 同样围绕 CUDA graph capture 的启动期行为做控制与观测，与本 PR 的捕获阶段 profiling 可相互配合排查启动期问题。
- PR #33775 [diffusion] feat: capture-safe pynccl all-to-all: 为目标 DiT 全前向 CUDA Graph 捕获做准备，说明仓库内多模块正在推进「捕获阶段可观测 / 可捕获」的工程化，与本 PR 方向一致。