

# PR #39176 完整报告

sgl-project/sglang

Reuse live CUDA graph executables during dedup registration

合并时间: 2026-09-13 02:44

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/39176>

## 执行摘要

- 一句话: 去重注册复用 CUDA live executable, 省显存并扩大共享范围
- 推荐动作: 值得精读。该 PR 体现了两个可借鉴的设计决策: 一是 " 签名粗筛 + 运行时精验 " 的分层校验思想——静态签名只做低成本粗分类, 精确兼容性交给 `cudaGraphExecUpdate` 在关键路径上验证; 二是信任并利用底层 API 的原子性语义 (失败的 update 不修改目标), 从而安全地去掉冗余保护实例。对需要维护 CUDA graph 捕获 / 执行机制、或设计去重 / 缓存类系统的工程师尤其有参考价值。

## 功能与动机

PR body 明确说明问题与方案: "Dedup registration instantiates an extra executable and separates compatible captures by grid size. Fix: Validate updates against the live executable and omit grid dimensions and redundant kernel names from signatures. Failed updates preserve the executable." 即原有 dedup 注册为了安全验证会多实例化一个 executable, 且 grid 维度差异阻碍了本可共享的捕获图合并, 造成不必要的显存与启动开销。

## 实现拆解

本 PR 变更集中在 `python/sglang/srt/model_executor/runner_backend/cuda_graph_dedup_mixin.py`, 并新增对应单测。拆解如下:

1. 签名瘦身 (数据契约变更): `kernel_node_payload()` 从签名载荷中移除 `kernel_name(params)` 字符串与 `(gridDimX, gridDimY, gridDimZ)`。kernel 名称可由 `kern/func handles` 隐含, 无需单独解析; grid 维度在不同 bucket 间属于可更新差异, 兼容性交给 `cudaGraphExecUpdate` 在 `register/replay` 时验证。保留 `handles` 的原因是代码注释中记录的 `sgl-project/sglang#37657` 驱动缺陷: `cuGraphKernelNodeGetAttribute` 读不回 `preferred cluster dimension`, 而 `cudaGraphExecUpdate` 在该字段变化时会错误报告成功, 待驱动修复后 `handles` 也可移除。
2. 数据结构简化: `GraphExecGroup` 删除 `compat_exec: int | None` 字段; `seal()` 与 `close()` 中对应的 `destroy_exec(compat_exec)` 清理逻辑一并删除, `seal()` 变成仅置 `sealed = True` 的幂等操作。
3. register 复用 live executable: `register()` 在签名命中 group 时, 直接对 `group.graph_exec` (真正用于 replay 的 live executable) 调用 `dedup_update()` 做兼容性验证, 成功后同步 `group.current_raw_graph = graph.raw_graph`, 使紧随其后的首次

replay 免于重复 update；不再对临时 compat\_exec 验证后再决定归属。注释明确依赖 CUDA 语义：失败的 cudaGraphExecUpdate 不会修改目标 executable。

4. replay 路径不变：replay() 仍保留按需 update 逻辑（current\_raw\_graph != raw\_graph 时更新后 launch），因此同一 group 内多个图交替 replay 的行为不受影响。
5. 测试配套：新增 test/registered/kernel/cuda\_graph/test\_cuda\_graph\_dedup.py（76 行），注册到 base-b-kernel-unit CI 阶段。test\_grid\_sizes\_share\_one\_executable 用 patch.object(registry, "instantiate", wraps=...) 断言两次不同 size 的注册只触发 1 次实例化，并验证两个图交替 replay 结果正确；test\_rejected\_registration\_preserves\_live\_executable 用 mock 强制签名相等后注册一个多节点的不兼容图，断言 AssertionError 抛出后 live executable 仍可正常 replay 原图。

关键文件：

- python/sglang/srt/model\_executor/runner\_backend/cuda\_graph\_dedup\_mixin.py（模块图去重；类别 source；类型 core-logic；符号 kernel\_node\_payload, register, replay, seal）：核心源码变更：删除 kernel\_name 函数，从签名载荷中移除 grid 维度与 kernel 名称；GraphExecGroup 删除 compat\_exec 字段；register 改为直接对 live executable 做 cudaGraphExecUpdate 验证并同步 current\_raw\_graph；seal/close 清理逻辑简化。
- test/registered/kernel/cuda\_graph/test\_cuda\_graph\_dedup.py（模块图去重；类别 test；类型 test-coverage；符号 TestCudaGraphDedup, capture, test\_grid\_sizes\_share\_one\_executable, test\_rejected\_registration\_preserves\_live\_executable）：新增的专项测试文件，覆盖两个关键行为：不同 grid size 的捕获图共享单个 executable（验证签名瘦身生效），以及注册失败时 live executable 不被破坏（验证对 CUDA 驱动语义的依赖）。

关键符号：kernel\_node\_payload, register, replay, seal, close, kernel\_name

## 关键源码片段

python/sglang/srt/model\_executor/runner\_backend/cuda\_graph\_dedup\_mixin.py

核心源码变更：删除 kernel\_name 函数，从签名载荷中移除 grid 维度与 kernel 名称；GraphExecGroup 删除 compat\_exec 字段；register 改为直接对 live executable 做 cudaGraphExecUpdate 验证并同步 current\_raw\_graph；seal/close 清理逻辑简化。

```
def kernel_node_payload(node):
    """提取 kernel 节点的签名载荷，用于 dedup group 的归属判断。"""
    assert cuda_drv is not None
    params = checkCudaErrors(cuda_drv.cuGraphKernelNodeGetParams(node))
    # grid 维度在不同 bucket（不同 batch size）之间会变化，属于可更新差异，
    # 不再写入签名，兼容性改由 cudaGraphExecUpdate 在 register / replay 时验证。
    # kernel 名称由 kern / func handles 隐含，单独读 name 属于冗余工作；
    # 仍保留 handles 是因为 cuGraphKernelNodeGetAttribute 读不回
    # preferred cluster dimension，而 cudaGraphExecUpdate 在该字段变化时
    # 会错误地报告成功（见 sgl-project/sglang#37657），
    # 待驱动修复后 handles 也可从签名中移除。
    return (
```

```

(int(params.kern), int(params.func)),
(int(params.blockDimX), int(params.blockDimY), int(params.blockDimZ)),
int(params.sharedMemBytes),
kernel_attrs(node),
)

```

```

class DedupedCudaGraphRegistry:

```

```

    # ... 前置逻辑略 ...

```

```

def register(self, captured_graph) -> DedupedCudaGraph:

```

```

    """注册捕获图：签名命中则复用 live executable，未命中则新建 group。"""

```

```

    assert not self.sealed

```

```

    raw_graph = captured_graph.raw_cuda_graph()

```

```

    signature = graph_signature(raw_graph)

```

```

    graph = DedupedCudaGraph(raw_graph, captured_graph, self)

```

```

    group = self.groups.get(signature)

```

```

    if group is not None:

```

```

        # CUDA 保证失败的 cudaGraphExecUpdate 不会修改目标 executable,

```

```

        # 所以可以直接用 live executable 做兼容性验证，无需临时实例。

```

```

        ok, detail = dedup_update(group.graph_exec, graph.raw_graph)

```

```

        assert ok, f"CUDA graph dedup register update failed ({detail})"

```

```

        # 同步当前内容，使紧随其后的首次 replay 免于重复 update。

```

```

        group.current_raw_graph = graph.raw_graph

```

```

        graph.group = group

```

```

        group.graphs.append(graph)

```

```

        return graph

```

```

    # 首个匹配：实例化一次 live executable，group 内所有 graph 共用。

```

```

    group = GraphExecGroup(

```

```

        graph_exec=self.instantiate(graph.raw_graph),

```

```

        current_raw_graph=graph.raw_graph,

```

```

        graphs=[graph],

```

```

    )

```

```

    graph.group = group

```

```

    self.groups[signature] = group

```

```

    return graph

```

```

def replay(self, graph: DedupedCudaGraph, stream: int) -> None:

```

```

    """按需把 live executable 更新到目标捕获图，再 launch。"""

```

```

    assert cuda_rt is not None

```

```

    group = graph.group

```

```

    assert group is not None, (

```

```

        "captured CUDA graph does not belong to this dedup state"

```

```

    )

```

```

    raw_graph = graph.raw_graph

```

```

    graph_exec = group.graph_exec

```

```

    if group.current_raw_graph != raw_graph:

```

```

    ok, detail = dedup_update(graph_exec, raw_graph)
    assert ok, (
        "CUDA graph dedup replay update failed "
        f"({detail}); captured graph is not compatible with its dedup group"
    )
    group.current_raw_graph = raw_graph

```

```

checkCudaErrors(cuda_rt.cudaGraphLaunch(graph_exec, stream))

```

## test/registered/kernel/cuda\_graph/test\_cuda\_graph\_dedup.py

新增的专项测试文件，覆盖两个关键行为：不同 grid size 的捕获图共享单个 executable（验证签名瘦身生效），以及注册失败时 live executable 不被破坏（验证对 CUDA 驱动语义的依赖）。

```

def test_grid_sizes_share_one_executable(self):
    """不同 grid size 的捕获图应共享同一个 executable。"""
    registry = dedup.DedupedCudaGraphRegistry()
    self.addCleanup(registry.close)
    captures = [self.capture(size) for size in (4096, 8192)]
    # grid 维度已从签名中去掉，两个图的签名应当一致。
    self.assertEqual(
        dedup.graph_signature(captures[0][0].raw_cuda_graph()),
        dedup.graph_signature(captures[1][0].raw_cuda_graph()),
    )
    # 只允许实例化一次：第二个注册必须复用第一个的 executable。
    with patch.object(
        registry, "instantiate", wraps=registry.instantiate
    ) as instantiate:
        graphs = [registry.register(capture[0]) for capture in captures]
    self.assertEqual(instantiate.call_count, 1)
    self.assertEqual(registry.stats(), (2, 1))
    self.assertEqual(graphs[0].group.current_raw_graph, graphs[1].raw_graph)
    # 两个图交替 replay，行为一致。
    registry.seal()
    for value in (3, 7):
        for graph, (_, inputs, outputs) in zip(graphs, captures):
            inputs.fill_(value)
            graph.replay()
            torch.cuda.synchronize()
            self.assertTrue(
                torch.equal(outputs, torch.full_like(outputs, value + 1))
            )

def test_rejected_registration_preserves_live_executable(self):
    """不兼容的注册被拒绝后，live executable 必须保持可用。"""
    registry = dedup.DedupedCudaGraphRegistry()
    self.addCleanup(registry.close)
    original, inputs, outputs = self.capture(4096)
    incompatible, _, _ = self.capture(4096, extra_node=True)

```

```

self.addCleanup(incompatible.reset)
graph = registry.register(original)
signature = dedup.graph_signature(graph.raw_graph)
# 强制签名一致，迫使 update 走失败分支（多一个节点导致不兼容）。
with (
    patch.object(dedup, "graph_signature", return_value=signature),
    self.assertRaisesRegex(AssertionError, "register update failed"),
):
    registry.register(incompatible)
# 失败后仍只有 1 个 group、1 个 graph，live executable 内容未变。
self.assertEqual(registry.stats(), (1, 1))
self.assertEqual(graph.group.current_raw_graph, graph.raw_graph)
inputs.fill_(11)
graph.replay()
torch.cuda.synchronize()
self.assertTrue(torch.equal(outputs, torch.full_like(outputs, 12)))

```

## 评论区精华

本 PR 无实质性 review 评论线程，唯一的 Issue 评论是作者触发 [/tag-and-rerun-ci](#)。设计权衡主要体现在代码注释与 PR body 中：

- 作者在 PR body 中声明 "Failed updates preserve the executable"，这是整个方案成立的前提——直接对 live executable 验证而不必保留单独的 compat\_exec。
- 代码注释记录了与 [sgl-project/sglang#37657](#) 相关的驱动行为缺陷：preferred cluster dimension 变化时 cudaGraphExecUpdate 会谎报成功，因此签名中暂时保留 kern/func handles 作为保险，等驱动修复后再移除。
- 测试文件通过 mock 强制签名一致来覆盖 update 失败路径，说明作者对“签名只是粗筛、运行时 update 才是精验”这一分工有明确认识。
- 暂无高价值评论线程

## 风险与影响

- 风险：
  1. 依赖 CUDA 驱动语义：方案成立的前提是 "不兼容的 cudaGraphExecUpdate 不修改 executable"，这是 CUDA 文档保证的行为，且新测试覆盖了失败路径；但若驱动出现偏差，live executable 可能处于不确定状态，后续 replay 会出错。目前失败路径是 AssertionError 硬失败，不会静默降级。
  2. 签名契约变更的影响面：签名移除 grid 维度与 kernel 名称后，dedup group 的合并范围显著扩大，原先因 grid size 不同而拆分的 bucket 现在共享同一 executable；replay 时跨捕获图切换仍依赖已有的按需 update 机制，频繁切换场景下 cudaGraphExecUpdate 的调用开销可能上升。
  3. 驱动缺陷残留风险：[sgl-project/sglang#37657](#) 中 preferred cluster dimension 的误判问题尚未解决，签名保留 handles 只是缓解手段，不兼容场景仍可能被误判为兼容。

4. 平台覆盖不全: PR 仅在 GB300 / PyTorch 2.11 / CUDA 13.0 实测; CI 中 AMD ROCm 10 运行失败 (带 `bypass-fastfail` 标签合入), 本变更涉及 `cuda_drv / cuda_rt` 路径, ROCm 平台行为未验证。 - 影响: 对用户: 每个 dedup group 的 CUDA graph executable 数量从 2 个降为 1 个, 且不同 grid size 的 bucket 可共享, 整体显存占用与注册阶段实例化耗时下降, 对 bucket 数量多的长上下文或大 batch 场景收益明显。对系统: `cuda_graph_dedup_mixin.py` 是 CUDA graph 执行后端的关键路径, 签名结构 (数据契约) 变化会影响所有使用 dedup 的模型执行; 但 `register/replay` 对外接口不变, `DedupedCudaGraph` 的用法 (`register`、`replay`、`seal`、`close`) 保持兼容。对团队: 新增测试固化了 "更新失败不破坏 live executable" 这一关键保证, 后续修改该机制时回归成本降低。 - 风险标记: 核心执行路径变更, 依赖 CUDA 驱动语义, 签名契约变更, AMD CI 未验证

## 关联脉络

- PR #39177 Scope graph-pool borrowing to the runtime and reduce fragmentation: 同一作者对 `cuda_graph_dedup_mixin.py` 的近期改动, 属于 CUDA graph 去重与 graph-pool 借用机制的连续演进; 本 PR 进一步减少 dedup group 的 executable 数量, 与 39177 的碎片化治理目标一致。
- PR #37657 (issue 引用) CUDA graph dedup 相关驱动行为缺陷: 代码注释明确引用 `sgl-project/sglang#37657` 作为保留 kern/func handles 的原因: preferred cluster dimension 变化时 `cudaGraphExecUpdate` 误报成功。