PR #33352 完整报告

sgl-project/sglang

fix: always capture default prefill CUDA graph

合并时间：2026-08-08 19:24

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/33352

执行摘要

- 一句话：移除 prefill CUDA graph 内存门槛，修复默认配置静默性能回退
- 推荐动作：值得精读。该 PR 展示了一个典型的设计决策反转：用配置驱动取代硬编码安全门槛，并伴随测试策略的同步调整（删除阈值单测、新增低显存行为回归）。阅读时可关注 capture_prefill_graph 的守卫顺序与日志信息，并结合 #31204 的实测数据判断默认行为与启动可靠性的平衡是否合理。

功能与动机

PR body 明确指出根因："The memory gate used a model-agnostic fixed threshold after weights, KV cache, and eager buffers were allocated. That allowed default configurations to disable prefill CUDA graphs at runtime and introduced a silent performance regression for otherwise supported models." 作者主张显存大小应由缓存 / 捕获参数（如 mem-fraction-static）负责，自动选择的 prefill graph 不应被硬编码空闲显存阈值覆盖；同时这是对 #31204 及其 review 反馈的后续修正。

实现拆解

1. 移除内存门槛常量与谓词：在 python/sglang/srt/model_executor/model_runner_components/cuda_graph_setup.py 中删除 _MIN_AUTO_PREFILL_CUDA_GRAPH_FREE_MEMORY_GB = 4.0 与 should_skip_auto_prefill_cuda_graph_for_memory()，并移除 capture_prefill_graph 中的调用分支与防御性 getattr(model_runner.server_args, "_cuda_graph_config_locked", set())。原因是固定阈值与模型无关，会在权重、KV cache、eager 缓冲区分配后误伤本可支持的模型，造成静默 eager 回退。
2. 简化捕获路径：capture_prefill_graph 现在无论剩余显存多少都尝试构造 PrefillCudaGraphRunner，空闲显存仅记录在日志 before_mem / after_mem 中，用于统计图捕获内存占用 mem_usage 与耗时 capture_time；显式后端检查 check_cuda_graph_backend、模型无 layers 属性、非标准 GQA 等兼容性守卫原样保留。
3. 测试替换与回归覆盖：删除 test_cuda_graph_setup.py 中 test_auto_prefill_cuda_graph_memory_gate 与 test_explicit_prefill_backend_bypasses_memory_gate 及其 Phase 导入；在 test_prefill_cuda_graph_runner.py 新增 test_low_free_memory_still_captures_prefill_graph，通过 patch.object(graph_setup, "get_available_gpu_memory", side_effect=[3.99, 3.5]) 模拟捕获前 3.99 GiB 空闲，断言最终 runner 是

PrefillCudaGraphRunner 而非 eager 回退。

4. 冒烟验证：PR 作者在 H100 上用默认参数 sglang serve --model-path Qwen/Qwen3-8B 验证：解析到 prefill.backend='breakable'，58 个 prefill bucket 全部捕获，耗时 26.30 s、占用 1.74 GiB，服务 health-ready 且 warmup 日志显示 cuda graph: True；本地 pytest 5 个用例与 2 个子测试通过。

关键文件：

- python/sglang/srt/model_executor/model_runner_components/cuda_graph_setup.py（模块 图捕获；类别 source；类型 data-contract；符号 should_skip_auto_prefill_cuda_graph_for_memory）：核心变更文件：移除 4 GiB 固定门槛常量、should_skip_auto_prefill_cuda_graph_for_memory 谓词及其调用分支，使自动 prefill CUDA graph 捕获不再受空闲显存硬编码阈值阻断。
- test/registered/unit/model_executor/test_prefill_cuda_graph_runner.py（模块 图测试；类别 test；类型 test-coverage；符号 test_low_free_memory_still_captures_prefill_graph）：新增回归测试 test_low_free_memory_still_captures_prefill_graph，验证 3.99 GiB 空闲显存下仍走 PrefillCudaGraphRunner 捕获路径，直接覆盖旧门槛边界。
- test/registered/unit/model_executor/model_runner_components/test_cuda_graph_setup.py（模块 图测试；类别 test；类型 test-coverage；符号 test_auto_prefill_cuda_graph_memory_gate, test_explicit_prefill_backend_bypasses_memory_gate）：清理过时测试：删除针对旧 4 GiB 阈值的两个单测与 Phase 导入，与新行为保持一致。

关键符号：capture_prefill_graph, should_skip_auto_prefill_cuda_graph_for_memory, test_low_free_memory_still_captures_prefill_graph

关键源码片段

python/sglang/srt/model_executor/model_runner_components/cuda_graph_setup.py

核心变更文件：移除 4 GiB 固定门槛常量、should_skip_auto_prefill_cuda_graph_for_memory 谓词及其调用分支，使自动 prefill CUDA graph 捕获不再受空闲显存硬编码阈值阻断。

```
# capture_prefill_graph 的核心捕获段（head 版本）。
# 前置的显式禁用与模型兼容性守卫（如模型缺少 layers 属性、非标准 GQA）
# 仍然存在，此处省略；被删除的是基于空闲显存的 4 GiB 硬编码门槛。
def capture_prefill_graph(*, model_runner: ModelRunner) -> GraphCapture:
    tic = time.perf_counter()
    # 捕获前的空闲显存只用于日志与内存占用统计，不再参与是否捕获的决策。
    before_mem = get_available_gpu_memory(model_runner.device, model_runner.gpu_id)
    role = "draft" if model_runner.is_draft_worker else "target"
    capture_name = f"{role} prefill"
    logger.info(
        f"Capture {capture_name} CUDA graph begin. "
        f"backend={prefill_backend}, num_tokens={capture_num_tokens}, "
        f"avail mem={before_mem:.2f} GB"
    )
```

```
    # 无论剩余显存多少，都尝试构造 multi-bucket prefill 图；
    # 显存容量由缓存 / 捕获参数（如 mem-fraction-static）负责控制。
    prefill_runner = PrefillCudaGraphRunner(model_runner)

    # 捕获结束后统计实际消耗，写入 GraphCapture 供上层汇总显存与耗时。
    after_mem = get_available_gpu_memory(model_runner.device, model_runner.gpu_id)
    mem_usage = before_mem - after_mem
    capture_time = time.perf_counter() - tic
    logger.info(
        f"Capture {capture_name} CUDA graph end. "
        f"elapsed={capture_time:.2f} s, "
        f"mem usage={mem_usage:.2f} GB, avail mem={after_mem:.2f} GB."
    )
    return result(prefill_runner, mem_usage, capture_time)
```

test/registered/unit/model_executor/test_prefill_cuda_graph_runner.py

新增回归测试 test_low_free_memory_still_captures_prefill_graph，验证 3.99 GiB 空闲显存下仍走 PrefillCudaGraphRunner 捕获路径，直接覆盖旧门槛边界。

```
# 回归测试：3.99 GiB 空闲显存时，自动捕获仍应继续（旧门槛已移除）。
# 这是对 #31204 引入的阈值行为的反向验证。
def test_low_free_memory_still_captures_prefill_graph(self):
    eager_runner = object()
    prefill_runner = object()
    model_runner = SimpleNamespace(
        device="cuda",
        gpu_id=0,
        is_draft_worker=False,
        spec_algorithm=SimpleNamespace(is_eagle=lambda: False),
        server_args=SimpleNamespace(
            enable_lora=False,
            cuda_graph_config=SimpleNamespace(
                prefill=SimpleNamespace(bs=[1], backend=Backend.BREAKABLE)
            ),
        ),
        model=SimpleNamespace(),
        model_config=SimpleNamespace(context_len=8192, num_hidden_layers=1),
        req_to_token_pool=SimpleNamespace(size=1),
    )
    # 模拟一个包含 layers 属性的模型，通过兼容性守卫
    language_model = SimpleNamespace(layers=[object()])

    with (
        patch.object(graph_setup, "check_cuda_graph_backend", return_value=False),
        patch.object(graph_setup, "resolve_language_model", return_value=language_model),
        patch.object(
            graph_setup,
            "compute_attention_and_moe_layers",
            return_value=([object()], [], [], [], []),
```

```
        ),
        # side_effect 两次取值：捕获前 3.99 GiB（低于旧阈值）、捕获后 3.5 GiB
        patch.object(
            graph_setup,
            "get_available_gpu_memory",
            side_effect=[3.99, 3.5],
        ),
        patch.object(
            graph_setup,
            "PrefillCudaGraphRunner",
            return_value=prefill_runner,
        ),
    ):
        capture = capture_prefill_graph(
            model_runner=model_runner,
            eager_runner=eager_runner,
        )

    # 关键断言：返回的是 prefill runner 而不是 eager runner
    self.assertIs(capture.runner, prefill_runner)
```

评论区精华

本 PR 无线上 review 评论可提炼，核心讨论来自 PR body 与关联 issue #31204 的设计权衡：

- 31204 立场：捕获前空闲显存不足 4 GiB 时，多 bucket 自动捕获会 OOM 或无法推进（Kimi-K2.7-Code TP=8 实测仅 2.05–2.28 GiB/rank 空闲），因此需要保守门槛保护启动可靠性。
- 本 PR 立场：模型无关固定阈值会在默认配置下静默禁用 prefill CUDA graph，造成受支持模型性能回归；显存容量应由 mem-fraction-static 等配置参数控制，显式 prefill backend 仍可覆盖默认行为。
- 最终决策：采用本 PR 方案，移除门槛与防御性 getattr，保留显式后端覆盖与模型兼容性守卫，并以 3.99 GiB 回归测试确认低显存仍捕获。
- 固定内存门槛是否应从自动捕获路径移除 (design): 合并方采纳本 PR 方案：移除门槛与防御性getattr，保留显式后端覆盖和模型兼容性守卫；3.99GiB回归测试确认低显存仍捕获。

风险与影响

- 风险：
  1. 低显存启动风险回归：#31204 修复的场景（Kimi-K2.7-Code，TP=8，--mem-fraction-static 0.97，捕获前每 rank 仅 2.05–2.28 GiB 空闲）在移除门槛后可能再次出现自动捕获 OOM 或启动无法推进；capture_prefill_graph 现在无条件构造 PrefillCudaGraphRunner，失败时不再优雅回退 eager。
  2. 回归测试盲区：新测试用单 rank 3.99 GiB 模拟“不跳过”，未覆盖真实低显存大模型多 rank 启动路径，无法验证最坏场景是否会 OOM。

3. 影响面：仅影响 prefill 自动捕获路径，*decode* 图捕获与显式 prefill backend 覆盖不受影响；用户仍可通过显式 *backend* 或降低 *mem-fraction-static* 规避。 - 影响：对默认部署，原先因 4 GiB 门槛被静默禁用 prefill CUDA graph 的模型会重新启用自动捕获，prefill 吞吐与首 token 延迟预期改善（H100 冒烟显示 58 个 bucket 全部捕获、仅 1.74 GiB 开销）；对高显存占用配置（大模型 + 大 KV *cache* + 高 *mem-fraction-static*）则引入启动失败风险回归。对团队而言，明确了“显存容量由配置参数负责、捕获路径不做硬编码门槛”的设计原则，测试资产也随之从阈值断言转向行为断言。 - 风险标记：低显存启动风险回归，缺少真实低显存场景回归测试，核心启动路径变更

关联脉络

- PR #31204 fix: skip *unsafe automatic prefill graph capture*: 本 PR 的直接前身，移除了其引入的 4 GiB 空闲显存门槛并替换对应测试。
- PR #32785 fix: *avoid piecewise prefill graph for trtllm_mla*: 同为 prefill CUDA graph 捕获路径的修复，涉及 server_args 与 *piecewise* 图注册逻辑。
- PR #34100 [Fix] *Give the piecewise CUDA graph test stub an hf_config*: 同为 prefill / *piecewise* CUDA graph 相关测试维护，说明该区域测试基础设施仍在演进。