PR #23351 完整报告

sgl-project/sglang

Support piecewise CUDA graph with NSA

合并时间：2026-05-23 05:39

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/23351

执行摘要

- 一句话：为 GLM-5/DSV3.2 添加 NSA 注意力 PCG 支持
- 推荐动作：值得精读。核心设计（register_split_op + register_custom_op 拆分 NSA 索引器）是 PCG 支持 DSA 模型的关键模式，可以推广到其他不符合 PCG 约束的算子。同时关注后续 PR #26718 对 guard 的改动，以及是否有更通用的 NSA indexer 抽象。

功能与动机

GLM-5/DSV3.2 currently doesn't allow piecewise CUDA graph due to incompatibilities in NSA attention backend and NSA indexer.

实现拆解

1. 注册自定义算子：在 dsa_indexer.py 中创建 k_cache_and_topk_result 和 logits_head_gate_pcg，分别用 @register_custom_op 和 @register_split_op 装饰，使 PCG 可以捕获 NSA 索引器的 store_index_k_cache 和 get_topk_ragged 操作。在 layernorm.py 和 hadamard.py 中为 FlashInfer layernorm 和 JIT Hadamard 变换注册类似算子（含 fake_impl 用于形状推断）。
2. 修复 torch.compile 兼容性：修改 _update_rope_guarded 中的 data_ptr() 比较，添加 not torch.compiler.is_compiling() 条件，避免在编译时触发。
3. 调整调度与配置：在 dsa_backend.py 的 set_dsa_prefill_impl 中，在 PCG 模式下强制关闭 MHA 分支（self.use_mha = False），因 PCG 无法动态分支。在 piecewise_context_manager.py 中添加 dsa_indexers 上下文传递，在 model_runner.py 和 piecewise_cuda_graph_runner.py 中设置。移除 server_args.py 中原有的 PCG 禁用 guard（此操作后续有争议）。
4. 添加端到端测试：新建 test/registered/piecewise_cuda_graph/test_pcg_glm5_fp4.py，使用 GLM-5-FP4 模型（TP=4）在 --enforce-piecewise-cuda-graph 下运行 GSM8K 评估，验证 PCG 正确性。

关键文件：

- python/sglang/srt/layers/attention/dsa/dsa_indexer.py（模块 索引器；类别 source；类型 core-logic；符号 k_cache_and_topk_result, _logits_head_gate_pcg_fake_impl, logits_head_gate_pcg）：核心变更文件，新增 k_cache_and_topk_result 和 logits_head_gate_pcg 两个 PCG 自定义算子，使 NSA 索引器可被 PCG 捕获。同时修复

_update_rope_guarded 的 torch.compile 兼容性。改动量最大（+167/-35）。

- test/registered/piecewise_cuda_graph/test_pcg_glm5_fp4.py（模块 集成测试；类别 test；类型 test-coverage；符号 TestPCGGlm5Fp4, setUpClass, tearDownClass, test_gsm8k）：新增的端到端测试文件，验证 GLM-5-FP4 模型在 PCG 下的 GSM8K 准确性。是 PCG for NSA 功能的关键质量保障。
- python/sglang/srt/layers/layernorm.py（模块 归一化层；类别 source；类型 core-logic；符号 _layernorm_fake_impl, layernorm）：为 FlashInfer layernorm 注册 PCG 自定义算子（含 fake_impl），使得 PCG 可以推理形状并捕获该层。是支持 NSA 模型 PCG 的基础部分。
- python/sglang/jit_kernel/hadamard.py（模块 JIT 内核；类别 source；类型 core-logic；符号 _hadamard_transform_fake_impl）：为 JIT 编译的 Hadamard 变换注册 PCG 自定义算子，使 Hadamard 变换可被 PCG 捕获，是 NSA 索引器中的一个步骤。
- python/sglang/srt/layers/attention/dsa_backend.py（模块 注意力后端；类别 source；类型 dependency-wiring）：修改 set_dsa_prefill_impl：在 PCG 模式下强制关闭 MHA，避免因动态分支导致捕获失败。是 PCG 与 NSA 调度交互的关键点。
- python/sglang/srt/compilation/piecewise_context_manager.py（模块 编译上下文；类别 source；类型 core-logic；符号 set_dsa_indexers）：添加 dsa_indexers 上下文，使 PCG 各分段可以共享 DSA 索引器引用，并提供 set_dsa_indexers 接口。是整个 PCG 上下文管理的一部分。

关键符号：k_cache_and_topk_result, logits_head_gate_pcg, layernorm, hadamard_transform, set_dsa_prefill_impl, _update_rope_guarded

关键源码片段

python/sglang/srt/layers/attention/dsa/dsa_indexer.py

核心变更文件，新增 k_cache_and_topk_result 和 logits_head_gate_pcg 两个 PCG 自定义算子，使 NSA 索引器可被 PCG 捕获。同时修复 _update_rope_guarded 的 torch.compile 兼容性。改动量最大（+167/-35）。

```
from sglang.srt.compilation.compilation_config import register_split_op
from sglang.srt.utils.custom_op import register_custom_op

# 注册为一个自定义操作，该操作会被 PCG 捕获为单个图节点
# `mutates_args=["topk_result"]` 声明 topk_result 会被就地修改
# `@register_split_op()` 标明该操作包含多个子步骤，PCG 可以拆分
@register_custom_op(mutates_args=["topk_result"])
@register_split_op()
def k_cache_and_topk_result(
    layer_id: int,
    key: torch.Tensor, # [total_tokens, head_dim] 的 key 张量
    q_fp8: torch.Tensor, # 量化后的 query（FP8）
    weights: torch.Tensor,
    topk_result: torch.Tensor, # [total_tokens, topk] 输出张量
) -> None:
    assert _is_cuda, "piecewise CUDA graph only supported on CUDA"
```

```
    from sglang.srt.layers.attention.dsa.triton_kernel import act_quant

    # 从 PCG 全局上下文中获取 forward_batch 和索引器
    forward_batch = get_forward_context().forward_batch
    indexer = get_forward_context().dsa_indexers[layer_id]
    metadata = get_attn_backend().get_indexer_metadata(layer_id, forward_batch)

    # 由于 PCG 会为所有静态 token 张量填充 padding，此处只取实际 token 数
    extend_num_tokens = forward_batch.extend_num_tokens

    # 执行 KV cache 存储（写入 key 和 scale）
    indexer._store_index_k_cache(
        forward_batch=forward_batch,
        layer_id=layer_id,
        key=key[:extend_num_tokens],
        act_quant=act_quant,
        out_cache_loc=forward_batch.out_cache_loc[:extend_num_tokens],
    )
    # 执行 topk 检索（从 KV cache 中选出最相关的块）
    indexer._get_topk_ragged(
        False,
        forward_batch,
        layer_id,
        q_fp8[:extend_num_tokens],
        weights[:extend_num_tokens],
        metadata,
        topk_result, # 注意 topk_result 已根据批次填充，无需切片
    )
```

test/registered/piecewise_cuda_graph/test_pcg_glm5_fp4.py

新增的端到端测试文件，验证 GLM-5-FP4 模型在 PCG 下的 GSM8K 准确性。是 PCG for NSA 功能的关键质量保障。

```
import unittest
from types import SimpleNamespace
from sglang.srt.utils import kill_process_tree
from sglang.test.ci.ci_register import register_cuda_ci
from sglang.test.run_eval import run_eval
from sglang.test.test_utils import (
    DEFAULT_TIMEOUT_FOR_SERVER_LAUNCH,
    DEFAULT_URL_FOR_TEST,
    CustomTestCase,
    popen_launch_server,
)

# 注册到 CI：预计运行 900 秒，属于 base-c 阶段，使用 4 卡 B200
register_cuda_ci(est_time=900, stage="base-c", runner_config="4-gpu-b200")

GLM5_FP4_MODEL = "nvidia/GLM-5-NVFP4"
```

```
class TestPCGGlm5Fp4(CustomTestCase):
    """PCG prefill on GLM-5-NVFP4 (DSA model, TP=4, B200)."""

    @classmethod
    def setUpClass(cls):
        # 启动服务器：TP=4，强制启用 PCG，启用模型加载多线程加速
        cls.process = popen_launch_server(
            GLM5_FP4_MODEL,
            DEFAULT_URL_FOR_TEST,
            other_args=[
                "--tp-size", "4",
                "--trust-remote-code",
                "--reasoning-parser", "glm45",
                "--tool-call-parser", "glm47",
                "--quantization", "modelopt_fp4",
                "--disable-flashinfer-autotune",
                "--enforce-piecewise-cuda-graph",
                "--model-loader-extra-config",
                '{"enable_multithread_load": true, "num_threads": 64}',
            ],
        )

    def test_gsm8k(self):
        # 运行 GSM8K 评估，200 条题目，期待准确率 > 0.92
        args = SimpleNamespace(
            base_url=DEFAULT_URL_FOR_TEST,
            model=GLM5_FP4_MODEL,
            eval_name="gsm8k",
            num_examples=200,
            num_threads=200,
            max_tokens=4096,
        )
        metrics = run_eval(args)
        self.assertGreater(metrics["score"], 0.92)
```

python/sglang/srt/layers/layernorm.py

为 FlashInfer layernorm 注册 PCG 自定义算子（含 fake_impl），使得 PCG 可以推理形状并捕获该层。是支持 NSA 模型 PCG 的基础部分。

```
if _is_flashinfer_available:
    try:
        import flashinfer.norm
        from sglang.srt.utils.custom_op import register_custom_op

        # 为 layernorm 定义一个 fake 实现，用于 PCG 的形状推断阶段
        # 只返回一个与输入形状相同的空张量，不执行实际计算
        def _layernorm_fake_impl(
            input: torch.Tensor,
            gamma: torch.Tensor,
```

```
        beta: torch.Tensor,
        eps: float = 1e-6,
    ) -> torch.Tensor:
        return torch.empty_like(input)

    # 注册为自定义操作，PCG 在捕获时遇到此函数会生成一个图节点
    # fake_impl 在形状推断阶段被调用，真实计算还是由 flashinfer 执行
    @register_custom_op(fake_impl=_layernorm_fake_impl)
    def layernorm(
        input: torch.Tensor,
        gamma: torch.Tensor,
        beta: torch.Tensor,
        eps: float = 1e-6,
    ) -> torch.Tensor:
        return flashinfer.norm.layernorm(input, gamma, beta, eps)

    _flashinfer_layernorm_available = True
except (ImportError, AttributeError):
    _flashinfer_layernorm_available = False
```

评论区精华

- server_args.py guard 争议：mmangkad 认为移除 guard 并不能保证所有 bypassed-topk MoE 内核在 PCG 下安全，并提交 #26718 恢复 guard。nvjullin 认为自定义 op 已充分修复。最终 PR 合并时 guard 被移除，后续需关注 mmangkad 的恢复 PR。
- 代码风格取舍：Fridge003 建议用 mixin 减少重复，nvjullin 反对，认为增加静态追踪难度，当前分支量可接受。双方达成一致维持现状。
- Tensor 切片位置：gemini-code-assist 建议在 k_cache_and_topk_result 中对 weights 也做切片，nvjullin 解释已在 radix_attention.py 统一处理，不需要。
- TRTLLM decode 输出维度：gemini-code-assist 发现 out 应该初始化为 4D，nvjullin 确认修复。
  - server_args.py 中 PCG guard 的移除是否安全 (design): PR 合并时 guard 被移除；mmangkad 提交了 #26718 恢复 guard，待后续处理。
  - NSA indexer 中过多 if _is_cuda 是否应重构 (style): 维持现状，不引入 mixin。
  - TRTLLM decode 输出张量维度错误 (correctness): nvjullin 确认修复，将 out 改为 4D 并在返回前 squeeze。
  - k_cache_and_topk_result 中 weights 是否也需要切片 (correctness): nvjullin 解释已在 radix_attention.py 统一处理，无需在此切片。

风险与影响

- 风险：
  1. CUDA 独占：PCG for NSA 仅支持 CUDA，其他后端（AMD、NPU）无法使用，且未添加显式回退或提示。

2. bypassed-topk MoE 兼容性：mmangkad 指出移除 guard 后，某些 bypassed-topk MoE 内核可能不安全，需后续 PR 显式豁免。
3. 自定义算子维护成本：register_custom_op 模式新增多个算子，需与 torch.compile/PCG 捕获逻辑同步更新，增加长期维护复杂度。
4. padding 处理依赖性：k_cache_and_topk_result 中的 extend_num_tokens 切片依赖准确的 forward_batch 状态，若未来修改 forward_batch 结构可能引入回归。 - 影响：影响范围：主要影响 GLM-5、DeepSeek V3.2 等使用 NSA 注意力的 DSA 模型用户，他们现在可以通过设置 --enforce-piecewise-cuda-graph 启用 PCG，获得推理延迟和吞吐量提升。非 DSA 模型不受影响。影响程度：对于目标模型，这是一个显著的性能优化（PR 中 benchmark 显示提升）。系统层面新增了 PCG 与 NSA 的交互路径，需要确保在 PCG 捕获和回退路径都正确工作。团队需要关注自定义算子的维护与后续兼容性。

- 风险标记：CUDA 独占，bypassed MoE 兼容性未充分验证，自定义算子维护成本，guard 移除争议

关联脉络

- PR #25983 (unknown - cause of merge conflict): 评论区提到该 PR 的合并导致 dsa_indexer.py 和 server_args.py 等文件的 bad merge，本 PR 修复了由此引发的回归。
- PR #26718 Restore PCG guard for non-DSA models: mmangkad 提出的 PR，旨在恢复 server_args.py 中被本 PR 移除的 guard，以确保非 DSA 模型不被意外启用 PCG。
- PR #26646 [core] Make overlap-schedule WAR barrier CUDA-only: 同属 PCG 相关领域（调度），但无直接代码冲突。可视为并行的 PCG 稳定性改进。