PR #29395 完整报告

sgl-project/sglang

[Spec] Capture DFLASH draft greedy sampling inside the draft decode cuda graph

合并时间：2026-06-28 10:34

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/29395

执行摘要

- 一句话：将 draft 贪婪采样并入 CUDA graph，提升 fwd_occupancy
- 推荐动作：值得精读，尤其关注 _DflashDraftSampler 的 capture-safe 设计以及 _maybe_build_draft_sampler 中的条件检查。该 PR 展示了如何在不破坏兼容性的前提下将 eager 操作融入 CUDA graph，是 CUDA graph 优化调用的良好范例。

功能与动机

PR body 指出：DFLASH 的 draft 模型没有自己的 LM head，借用 target 模型的 lm_head 进行贪婪采样。该操作之前以 eager 方式执行，产生额外 launch 且未被设备计时器计入 fwd_occupancy。通过折叠到 CUDA graph 中，能减少开销并准确计时，同时不改变 draft 行为。

实现拆解

1. 定义 _DflashDraftSampler 类(python/sglang/srt/speculative/dflash_worker_v2.py)：该类封装了 capture-safe 的贪婪 argmax 操作。构造函数接收 lm_head.weight、block_size 等参数，预分配输出缓冲区 self.out。__call__ 方法将 hidden_states 按 block 分割、转换为权重 dtype，执行 torch.matmul 和 torch.argmax，结果复制到输出缓冲区。
2. 构建 sampler 并附着到 model_runner(同一文件)：在 DFlashWorkerV2.init_cuda_graphs() 中，于调用 draft worker 的 init_cuda_graphs 之前，调用新增方法 _maybe_build_draft_sampler()。该方法检查条件：tp_rank==0、world_size==1、block_size>1、target 模型存在 lm_head 且权重为浮点型、未使用附加词汇表（added_vocab_size==0）。若满足条件则创建 _DflashDraftSampler 实例并赋值给 self.draft_model_runner.dflash_draft_sampler；否则返回 None（eager fallback）。
3. 在 CUDA graph capture 中调用 sampler(python/sglang/srt/model_executor/runner/decode_cuda_graph_runner.py)：在 capture_one 内部的 run_once 函数中，forward 调用之后通过 getattr 安全获取 model_runner.dflash_draft_sampler。如果存在且 out 是 LogitsProcessorOutput 且 hidden_states 不为 None，则调用 dflash_sampler(out.hidden_states)。若 hidden_states 缺失则抛出 RuntimeError，确保错误在 capture 阶段暴露。
4. 测试验证：无新测试文件，但 PR 通过 /rerun-test 触发了三个已有测试：test_dflash.py、test_pcg_with_speculative_decoding_dflash.py（1-gpu-5090）以及 test_gemma4_dflash_31b_extra.py（2-gpu-h100），全部通过。

关键文件：

- python/sglang/srt/speculative/dflash_worker_v2.py（模块 推测解码；类别 source；类型 core-logic；符号 _DflashDraftSampler, init, call, _maybe_build_draft_sampler）：引入 _DflashDraftSampler 类，构建并附着 sampler 到 model_runner，添加条件检查逻辑，是本次核心变更的载体。
- python/sglang/srt/model_executor/runner/decode_cuda_graph_runner.py（模块 解码器；类别 source；类型 data-contract）：在 CUDA graph capture 的 run_once 中增加 dflash sampler 调用，确保被捕获到 graph 中；添加运行时错误检查。

关键符号：_DflashDraftSampler.init, _DflashDraftSampler.call, DFlashWorkerV2._maybe_build_draft_sampler, DFlashWorkerV2.init_cuda_graphs, DecodeCudaGraphRunner.capture_one（内部 run_once）

关键源码片段

python/sglang/srt/speculative/dflash_worker_v2.py

引入 _DflashDraftSampler 类，构建并附着 sampler 到 model_runner，添加条件检查逻辑，是本次核心变更的载体。

```
class _DflashDraftSampler:
    """Capture-safe greedy argmax over the target LM head, run inside the draft
    cuda graph so the draft sampling is captured and counted in fwd_occupancy.
    DFLASH's draft has no head of its own; it borrows the target `lm_head`.
    tp=1 / no-added-vocab only; TP>1 stays eager in the worker.
    """

    def __init__(self, *, weight, block_size, num_org, org_vocab_start, max_bs):
        self.weight = weight
        self.block_size = int(block_size)
        self.num_org = int(num_org) # 原始词汇表大小（排除 added_vocab）
        self.org_vocab_start = int(org_vocab_start)
        # 输出缓冲区，写入 proposed draft tokens，在 replay 后由 worker 读取
        self.out = torch.empty(
            (int(max_bs) * (self.block_size - 1),),
            dtype=torch.int64,
            device=weight.device,
        )

    def __call__(self, hidden_states):
        # draft tokens 位于 block 位置 1:（位置 0 是 seeded bonus token）
        bs = hidden_states.shape[0] // self.block_size
        hs = hidden_states.view(bs, self.block_size, -1)[:, 1:, :].reshape(
            -1, hidden_states.shape[-1]
        )
        if hs.dtype != self.weight.dtype:
            hs = hs.to(self.weight.dtype)
        logits = torch.matmul(hs, self.weight[: self.num_org].T)
```

```
tokens = torch.argmax(logits, dim=-1).to(torch.long)
if self.org_vocab_start:
    tokens += self.org_vocab_start
self.out[: tokens.shape[0]].copy_(tokens)
```

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

在 CUDA graph capture 的 run_once 中增加 dflash sampler 调用，确保被捕获到 graph 中；添加运行时错误检查。

```
out = forward(
    forward_batch.input_ids,
    forward_batch.positions,
    forward_batch,
    **kwargs,
)
# 从 model_runner 获取 sampler（若为 None 则跳过）
dflash_sampler = getattr(
    self.model_runner, "dflash_draft_sampler", None
)
if dflash_sampler is not None:
    # 必须被捕获，否则 replay 时输出缓冲区可能残留无效 token —— 提前抛错
    if (
        not isinstance(out, LogitsProcessorOutput)
        or out.hidden_states is None
    ):
        raise RuntimeError(
            "DFLASH draft sampler set but the draft forward has no "
            "hidden_states to capture into the graph."
        )
    dflash_sampler(out.hidden_states)
return out
```

评论区精华

主要讨论围绕安全性和优化：

- gemini-code-assist[bot]指出需使用 getattr 避免 AttributeError，并检查缓冲区非 None 再进行切片，防止 TypeError。作者采纳，最终代码使用 getattr(self.model_runner, "dflash_draft_sampler", None)。
- 同一 reviewer建议用 torch.empty 替代 torch.zeros 避免零初始化开销，以及使用 += 替代 + 避免临时张量分配。最终实现均采用。
- 还建议检查 lm_head.weight 是否为浮点张量以防止量化权重导致 matmul 失败。最终代码在 _maybe_build_draft_sampler 中加入 torch.is_floating_point(lm_head.weight) 检查。

所有评论均被作者采纳并整合到最终提交中。

- 运行时安全检查：避免对 None 类型进行切片和属性访问 (correctness): 作者采纳，最终代码使用 getattr(self.model_runner, "dflash_draft_sampler", None) 并在调用 sampler 前检查其是否为 None。

- 性能优化：使用 torch.empty 避免零初始化开销 (performance): 作者采用，改为 torch.empty。
- 量化兼容性：检查 lm_head 权重是否为浮点型 (correctness): 作者采纳，在 _maybe_build_draft_sampler 中加入 torch.is_floating_point(lm_head.weight) 检查。

风险与影响

- 风险：主要风险：
  - CUDA Graph 捕获变更：修改了 DecodeCudaGraphRunner 的 run_once，在 forward 之后加入 sampler 调用。如果条件判断有误或 hidden_states 行为变化，可能导致 capture 失败或 replay 使用过期缓冲区。但已通过防御性 RuntimeError 和 getattr 缓解。
  - TP>1 与量化兼容性：通过 _maybe_build_draft_sampler 中的多重检查确保非 tp=1 或量化权重时回退到 eager 路径，不破坏现有流程。
  - 缺少新增测试：没有为新的 _DflashDraftSampler 单独编写单元测试，依赖已有集成测试覆盖。若未来重构可能被遗漏。
  - 性能影响：仅 tp=1 场景受益，tp>1 无变化。
  - 影响：对用户：tp=1 场景下 fwd_occupancy 提升约 5 个百分点，采样时间被正确计入。draft 行为无变化。对系统：CUDA graph capture 逻辑扩展，所有使用 DFLASH spec-v2 的请求均会经过新路径；TP>1 场景维持原有 eager 行为。对团队：设计模式可推广到其他需要将 eager 操作融入 CUDA graph 的场景，如 DSA（PR#29413）。
  - 风险标记：CUDA Graph 捕获变更，TP=1 特殊处理路径，eager fallback 兼容性，无新增测试文件

关联脉络

- PR #29413 [DSA] Enable draft-extend CUDA graph for DeepSeek Sparse Attention: 同样在 speculative decoding 中扩展 CUDA graph 覆盖范围，减少 eager launch，提升性能。虽算法不同，但思路相似：将之前 eager 执行的 draft 操作融入 CUDA graph。