PR #22342 完整报告

sgl-project/sglang

[AMD] Enable DFLASH speculative decoding on ROCm

合并时间：2026-04-18 04:10

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/22342

执行摘要

- 一句话：在 AMD ROCm 平台上启用 DFLASH 推测解码，支持 Triton 注意力后端。
- 推荐动作：该 PR 值得精读，特别是 dflash_worker.py 中的后端选择逻辑和 triton_backend.py 中的掩码防护设计，展示了如何优雅地处理平台差异和边缘情况。关注 ROCm 检测的实现方式及其对默认行为的影响。

功能与动机

根据 PR body 描述，DFLASH 推测解码目前仅支持 FlashInfer/FA3/FA4 后端，这些后端在 ROCm 平台上不可用。为了在 AMD ROCm GPU 上启用 DFLASH，需要添加 Triton 注意力后端作为草稿工作器的支持选项。

实现拆解

1. 扩展草稿工作器后端支持：在 dflash_worker.py 中，将 "triton" 添加到 supported_draft_backends 元组，并引入自动检测逻辑：若未指定后端或指定了不支持的后端，则根据 torch.version.hip 判断是否为 ROCm 环境，是则默认使用 "triton"，否则使用 "flashinfer"。这确保了 DFLASH 在 ROCm 上可用。
2. 修复 RoPE 内核形状兼容性：在 dflash.py 的 apply_k_rope 方法中，将 dummy_q 的形状从 (k.shape[0], self.head_dim) 改为与 k 相同的形状，以满足 sgl_kernel.rotary_embedding 对 num_heads % num_kv_heads == 0 的检查，避免在 ROCm（以及使用 sgl_kernel 的 CUDA 环境）上崩溃。
3. 防护自定义掩码访问：在 triton_backend.py 的 init_forward_metadata_capture_cuda_graph 和 init_forward_metadata_replay_cuda_graph 方法中，添加条件检查，仅在 spec_info.custom_mask 存在时才进行赋值，防止 DFLASH 使用非因果 ENCODER_ONLY 注意力（无自定义掩码）时在 CUDA 图捕获期间访问 None 属性导致崩溃。此修复对所有使用非因果注意力的推测模式都有益。
4. 模型层捕获支持：在 qwen3.py 中添加 set_dflash_layers_to_capture 方法（与 LlamaForCausalLM 中的模式相同），使 Qwen3 模型能够支持 DFLASH 的隐藏状态捕获。

关键文件：

- python/sglang/srt/speculative/dflash_worker.py（模块 推测解码；类别 source；类型 dependency-wiring；符号 init）：这是启用 DFLASH 在 ROCm 上支持的核心文件，通过添加 Triton 后端和自动检测逻辑，决定了草稿工作器的注意力后端选择。

- python/sglang/srt/layers/attention/triton_backend.py（模块 注意力层；类别 source；类型 core-logic；符号 init_forward_metadata_capture_cuda_graph, init_forward_metadata_replay_cuda_graph）：修复了在非因果注意力（如 DFLASH 使用的 ENCODER_ONLY）下，CUDA 图捕获期间访问 None 自定义掩码导致的崩溃，提升了稳定性。
- python/sglang/srt/models/dflash.py（模块 模型层；类别 source；类型 data-contract；符号 apply_k_rope）：修复了 RoPE 内核中的形状兼容性问题，确保在 ROCm 和特定 CUDA 环境下能正确运行，避免因头数检查失败而崩溃。

关键符号：init, apply_k_rope, init_forward_metadata_capture_cuda_graph, init_forward_metadata_replay_cuda_graph, set_dflash_layers_to_capture

关键源码片段

python/sglang/srt/speculative/dflash_worker.py

这是启用 DFLASH 在 ROCm 上支持的核心文件，通过添加 Triton 后端和自动检测逻辑，决定了草稿工作器的注意力后端选择。

```
# 在 DFlashWorker 的 __init__ 方法中，修改后端选择逻辑
supported_draft_backends = ("flashinfer", "fa3", "fa4", "triton") # 新增 "triton"
if draft_backend is None:
    draft_backend, _ = draft_server_args.get_attention_backends()
if draft_backend is None:
    # 在 ROCm 上使用 triton（无 FlashInfer），在 CUDA 上使用 flashinfer
    import torch as _torch # 注意：review 指出这是冗余导入，顶部已导入 torch
    draft_backend = "triton" if _torch.version.hip else "flashinfer"
elif draft_backend == "trtllm_mha":
    import torch as _torch # 重复导入
    _fb = "triton" if _torch.version.hip else "flashinfer"
    logger.warning(
        "DFLASH draft worker does not support 'trtllm_mha' because the "
        "draft path requires non-causal attention. Falling back to '%s'.",
        _fb,
    )
    draft_backend = _fb
elif draft_backend not in supported_draft_backends:
    import torch as _torch # 重复导入
    _fb = "triton" if _torch.version.hip else "flashinfer"
    logger.warning(
        "DFLASH draft worker only supports attention_backend in %s for now, "
        "but got %r. Falling back to '%s'.",
        supported_draft_backends,
        draft_backend,
        _fb,
    )
    draft_backend = _fb
# 使草稿工作器后端明确且自包含（无进一步覆盖）
draft_server_args.speculative_draft_attention_backend = None
```

```
draft_server_args.prefill_attention_backend = None
draft_server_args.decode_attention_backend = None
draft_server_args.attention_backend = draft_backend # 设置最终后端
```

python/sglang/srt/layers/attention/triton_backend.py

修复了在非因果注意力（如 DFLASH 使用的 ENCODER_ONLY）下，CUDA 图捕获期间访问 None 自定义掩码导致的崩溃，提升了稳定性。

```
# 在 init_forward_metadata_capture_cuda_graph 和 init_forward_metadata_replay_cuda_graph 方法中
custom_mask = self.cuda_graph_custom_mask
# 防护自定义掩码访问：仅在 spec_info 和 custom_mask 均存在时赋值
if (
    spec_info is not None
    and getattr(spec_info, "custom_mask", None) is not None
):
    custom_mask[: spec_info.custom_mask.shape[0]] = spec_info.custom_mask
else:
    custom_mask = None # 否则设为 None，避免后续使用时报错
# 此修复确保 DFLASH 等使用非因果注意力的模式在 CUDA 图捕获时不会崩溃
```

python/sglang/srt/models/dflash.py

修复了 RoPE 内核中的形状兼容性问题，确保在 ROCm 和特定 CUDA 环境下能正确运行，避免因头数检查失败而崩溃。

```
def apply_k_rope(self, positions: torch.Tensor, k: torch.Tensor) -> torch.Tensor:
    # 匹配 K 的形状，使 RoPE 内核的头数检查在所有后端上都能通过
    # 原注释：使用最小的虚拟查询（1个头）以避免完整 Q 工作
    # 新实现：dummy_q 形状与 k 相同，确保 num_heads % num_kv_heads == 0
    dummy_q = k.new_empty(k.shape) # 形状从 (k.shape[0], self.head_dim) 改为 k.shape
    _, k = self.rotary_emb(positions, dummy_q, k)
    return k
```

评论区精华

review 中，gemini-code-assist[bot] 指出 dflash_worker.py 中存在冗余的 torch 局部导入（文件顶部已导入）和重复的 ROCm 检测逻辑，建议将后备后端逻辑合并以提高可维护性。但 PR 最终合并时未采纳此建议，代码中仍保留了重复的导入和逻辑。hnyls2002 批准了 PR 并执行了合并。

- 代码冗余与可维护性 (style): PR 合并时未采纳建议，代码中保留了重复的导入和逻辑。

风险与影响

- 风险：1. 回归风险：dflash_worker.py 中新增的 ROCm 检测逻辑可能影响 CUDA 环境下的默认后端选择，若检测逻辑有误，可能导致 CUDA 上意外使用 Triton 后端，可能带来性能下降或兼容性问题。 2. 兼容性风险：dflash.py 的形状修复虽然解决了 ROCm 上的问题，但改变了 dummy_q 的张量形状，需确保所有后端（包括 FlashInfer、FA3、FA4）的 RoPE 内核都能正确处理新形状，否则可能引入新错误。 3. 代码质量风险：根据 review 评

论，dflash_worker.py 中存在冗余代码（重复导入和逻辑），可能降低代码可读性和维护性。

- 影响：1. 用户影响：AMD ROCm 平台的用户现在可以使用 DFLASH 推测解码来加速推理，扩展了 SGLang 在异构硬件上的功能覆盖。 2. 系统影响：增强了推测解码模块的硬件兼容性，使 DFLASH 成为跨 CUDA 和 ROCm 的统一解决方案。triton_backend.py 的修复也提升了其他推测模式在非因果注意力下的稳定性。 3. 团队影响：为后续在 AMD 平台上支持更多推测解码算法奠定了基础，减少了平台特定的代码分支。
- 风险标记：平台检测逻辑风险，形状兼容性变更，冗余代码

关联脉络

- PR #19545 feat(observability): add OpenTelemetry tracing for speculative decoding: 同属推测解码模块的功能增强，关注推测解码的可观测性，而本 PR 关注硬件平台支持。
- PR #22128 Allow piecewise CUDA graph with speculative decoding: 涉及推测解码与 CUDA 图的集成，本 PR 的 triton_backend.py 修复也涉及 CUDA 图捕获，有技术关联。
- PR #22952 [AMD] Add SGLANG_MORI_MOE_MAX_INPUT_TOKENS to truncate dispatch before MoE.: 同为 AMD 平台的功能扩展，关注性能优化和环境变量支持。