PR #49937 完整报告

vllm-project/vllm

[ROCm] Add AITER FP8 ViT encoder attention

合并时间：2026-07-30 14:22

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/49937

执行摘要

- 一句话：为 AMD MI300/MI350 添加 AITER FP8 ViT 注意力
- 推荐动作：值得精读，特别是 vllm/_aiter_ops.py 中的自定义 op 注册模式和 mm_encoder_attention.py 中后端子分发技巧，为未来添加更多 AMD 加速算子提供参考。测试和基准测试齐全，适合作为新硬件集成示例。

功能与动机

现有的 FP8 ViT 注意力路径仅支持 NVIDIA FlashInfer/cuDNN 后端，而 AMD ROCm 平台（MI300/MI350）无法利用 FP8 加速。本 PR 通过集成 AITER 库，为 AMD GPU 提供同等的 FP8 注意力计算能力，提升多模态编码器的推理性能。

实现拆解

1. 注册自定义 op：在 vllm/_aiter_ops.py 中实现 _rocm_aiter_fp8_attn_impl 和 fake 实现 _rocm_aiter_fp8_attn_fake，通过 direct_register_custom_op 注册为 aiter_fp8_attn_wrapper；为 rocm_aiter_ops 类添加静态方法 fp8_attn_wrapper 提供统一入口。
2. 添加 ViT 包装器：在 vllm/v1/attention/ops/vit_attn_wrappers.py 中新增 vit_aiter_fp8_attn_wrapper 函数，委托给 rocm_aiter_ops.fp8_attn_wrapper，作为 ViT 注意力包装器的新选项。
3. 扩展 MMEncoderAttention：在 vllm/model_executor/layers/attention/mm_encoder_attention.py 中修改 _init_fp8_state 方法，优先检测 ROCM_AITER_FA 后端并进行架构、模块验证；新增 _quantize_qkv_fp8 方法统一 FP8 量化逻辑；新增 _forward_aiter_fp8 方法动态 / 静态缩放模式下调用 AITER 包装器。
4. 测试覆盖：在 tests/kernels/attention/test_mha_attn.py 中添加 test_mha_attn_varlen_forward_aiter_fp8 验证变长输入正确性，test_mha_attn_aiter_fp8_rejects_unsupported_arch 验证不支持的架构被拒绝；在 tests/kernels/core/test_vit_fp8_scaling.py 中添加 test_aiter_static_scales_loaded 验证静态缩放路径。
5. 基准测试：新增 benchmarks/kernels/benchmark_vit_aiter_fp8_attn.py，使用 triton 的 do_bench 对比 BF16 和 FP8 路径性能，支持指定序列长度、头数量等参数。

6. 文档更新：修改 docs/features/quantization/fp8_vit_attn.md，增加 AITER FP8 的配置说明和用法示例。

关键文件：

- vllm/model_executor/layers/attention/mm_encoder_attention.py（模块 编码器注意力；类别 source；类型 core-logic；符号 _quantize_qkv_fp8, _forward_aiter_fp8）：核心注意力层，新增 AITER FP8 后端支持：修改 _init_fp8_state 控制流，新增 _quantize_qkv_fp8 和 _forward_aiter_fp8 方法，复用现有量化框架。
- vllm/_aiter_ops.py（模块 AITER 算子；类别 source；类型 core-logic；符号 _rocm_aiter_fp8_attn_impl, _rocm_aiter_fp8_attn_fake, fp8_attn_wrapper）：新增自定义 op 的实现和注册，包含 _rocm_aiter_fp8_attn_impl、fake 实现和 fp8_attn_wrapper 静态方法，是跨模块复用的统一入口。
- benchmarks/kernels/benchmark_vit_aiter_fp8_attn.py（模块 性能基准；类别 source；类型 test-coverage；符号 make_attention, bench）：新增基准测试脚本，清晰展示如何构建 AITER FP8 注意力并对比 BF16/FP16 性能，供用户复现和调优。
- vllm/v1/attention/ops/vit_attn_wrappers.py（模块 注意力包装器；类别 infra；类型 infrastructure；符号 vit_aiter_fp8_attn_wrapper）：新增 vit_aiter_fp8_attn_wrapper 作为 ViT 注意力包装器之一，将调用委托给 rocm_aiter_ops.fp8_attn_wrapper，保持与现有 wrapper 模式一致。
- tests/kernels/attention/test_mha_attn.py（模块 MHA 测试；类别 test；类型 test-coverage；符号 test_mha_attn_varlen_forward_aiter_fp8, test_mha_attn_aiter_fp8_rejects_unsupported_arch）：添加 AITER FP8 注意力核心测试：变长前向正确性、不支持架构拒绝，确保后端行为符合预期。
- tests/kernels/core/test_vit_fp8_scaling.py（模块 缩放测试；类别 test；类型 test-coverage；符号 test_aiter_static_scales_loaded）：新增 AITER 静态缩放测试，验证 FP8 缩放因子加载路径在 AITER 后端下正常工作。

关键符号：_rocm_aiter_fp8_attn_impl, _rocm_aiter_fp8_attn_fake, fp8_attn_wrapper, vit_aiter_fp8_attn_wrapper, _quantize_qkv_fp8, _forward_aiter_fp8, test_mha_attn_varlen_forward_aiter_fp8, test_mha_attn_aiter_fp8_rejects_unsupported_arch, test_aiter_static_scales_loaded, make_attention, bench

关键源码片段

vllm/model_executor/layers/attention/mm_encoder_attention.py

核心注意力层，新增 AITER FP8 后端支持：修改 _init_fp8_state 控制流，新增 _quantize_qkv_fp8 和 _forward_aiter_fp8 方法，复用现有量化框架。

```
def _init_fp8_state(self) -> None:
    # 省略初始化默认值的代码 ...
    mm_cfg = get_multimodal_config()
    if mm_cfg is None or mm_cfg.mm_encoder_attn_dtype != "fp8":
        return

    # 优先处理 AITER 后端
```

```
if self.attn_backend == AttentionBackendEnum.ROCM_AITER_FA:
    # 验证平台和架构
    if not current_platform.is_rocm():
        raise ValueError("AITER FP8 ViT attention requires ROCm.")
    from vllm.platforms.rocm import on_mi3xx
    if not on_mi3xx():
        raise ValueError(
            "AITER FP8 ViT attention requires an MI300-series or "
            "MI350-series GPU (gfx942 or gfx950)."
        )
    # 验证 AITER 模块可用
    try:
        from aiter import flash_attn_varlen_fp8_pertensor_func # noqa: F401
    except ImportError as exc:
        raise ValueError(
            "mm_encoder_attn_dtype='fp8' with ROCM_AITER_FA requires "
            "an AITER build that provides "
            "flash_attn_varlen_fp8_pertensor_func."
        ) from exc
elif self.attn_backend == AttentionBackendEnum.FLASHINFER:
    # 原有的 FlashInfer 检查
    if not is_flashinfer_cudnn_fp8_prefill_attn_supported():
        raise ValueError(...)
else:
    raise ValueError(...)
self.fp8_enabled = True
self._fp8_dynamic_scale = mm_cfg.mm_encoder_fp8_scale_path is None
self.fp8_quant = QuantFP8(static=True, group_shape=GroupShape.PER_TENSOR)
# 注册缩放缓冲区 ...
```

vllm/_aiter_ops.py

新增自定义 op 的实现和注册，包含 _rocm_aiter_fp8_attn_impl、fake 实现和 fp8_attn_wrapper 静态方法，是跨模块复用的统一入口。

```
def _rocm_aiter_fp8_attn_impl(
    q: torch.Tensor,
    k: torch.Tensor,
    v: torch.Tensor,
    q_descale: torch.Tensor,
    k_descale: torch.Tensor,
    v_descale: torch.Tensor,
    batch_size: int,
    output_dtype: torch.dtype,
    scale: float | None = None,
    cu_seqlens: torch.Tensor | None = None,
    max_seqlen: torch.Tensor | None = None,
) -> torch.Tensor:
    """调用 AITER 的变长 FP8 注意力函数。"""
    from aiter import flash_attn_varlen_fp8_pertensor_func
```

```
    q_len = q.size(1)
    if cu_seqlens is None:
        cu_seqlens = torch.arange(
            0, (batch_size + 1) * q_len, step=q_len, dtype=torch.int32, device=q.device
        )
    max_seqlen_value = q_len if max_seqlen is None else max_seqlen.item()

    # 展平批次和序列维度以适应 AITER 的 2D 输入
    q, k, v = (x.flatten(0, 1) for x in (q, k, v))
    output = flash_attn_varlen_fp8_pertensor_func(
        q, k, v,
        q_descale=q_descale, k_descale=k_descale, v_descale=v_descale,
        cu_seqlens_q=cu_seqlens, cu_seqlens_k=cu_seqlens,
        max_seqlen_q=max_seqlen_value, max_seqlen_k=max_seqlen_value,
        causal=False, softmax_scale=scale,
    )
    return output.to(output_dtype).reshape(batch_size, q_len, *output.shape[1:])

# 在 class rocm_aiter_ops 中注册注册后添加:
class rocm_aiter_ops:
    @staticmethod
    def fp8_attn_wrapper(...):
        # 委托给注册的 custom op
        return torch.ops.vllm.aiter_fp8_attn_wrapper(...)
```

benchmarks/kernels/benchmark_vit_aiter_fp8_attn.py

新增基准测试脚本，清晰展示如何构建 AITER FP8 注意力并对比 BF16/FP16 性能，供用户复现和调优。

```
def make_attention(num_heads: int, head_dim: int, fp8: bool) -> MMEncoderAttention:
    mm_config = MultiModalConfig(
        mm_encoder_attn_backend=AttentionBackendEnum.ROCM_AITER_FA,
        mm_encoder_attn_dtype="fp8" if fp8 else None,
    )
    vllm_config = VllmConfig()
    vllm_config.model_config = SimpleNamespace(multimodal_config=mm_config)
    with set_current_vllm_config(vllm_config):
        return MMEncoderAttention(num_heads, head_dim).to("cuda")

def bench(seq_lens, num_heads, head_dim, warmup_ms, repeat_ms):
    # 创建 FP8 和 BF16 注意力实例
    fp8_attn = make_attention(num_heads, head_dim, fp8=True)
    bf16_attn = make_attention(num_heads, head_dim, fp8=False)
    # 对每个序列长度进行预热和计时
    for seq_len in seq_lens:
        qkv = torch.randn(1, seq_len, 3, num_heads, head_dim).cuda()
        q, k, v = qkv.unbind(dim=2)
        cu_seqlens = torch.tensor([0, seq_len]).cuda()
        # 先校准动态缩放，再测量静态缩放路径
```

```
fp8_attn._fp8_dynamic_scale = True
fp8_attn._forward_aiter_fp8(q, k, v, cu_seqlens, ...)
fp8_attn._fp8_dynamic_scale = False
# 使用 triton.do_bench 测量
bf16_ms = triton.testing.do_bench(lambda: bf16_attn._forward_fa(q, k, v, ...), ...)
fp8_ms = triton.testing.do_bench(lambda: fp8_attn._forward_aiter_fp8(q, k, v, ...), ...)
print(f"{seq_len} {bf16_ms:.3f} {fp8_ms:.3f} {bf16_ms/fp8_ms:.2f}x")
```

评论区精华

review 中 tjtanaa 建议将自定义 op 从 vit_attn_wrappers.py 移至 vllm/_aiter_ops.py 以便跨模块复用，作者 LiuYinfeng01 采纳并完成了迁移，并确认迁移后性能一致（固定输入比率 1.0005，变长输入比率 0.9957）。此讨论已解决，无未完成疑虑。

- op 注册位置重构：从 vit_attn_wrappers.py 移至 _aiter_ops.py (design): 作者 LiuYinfeng01 同意并完成迁移，确认性能与原有实现一致。

风险与影响

- 风险：
  1. 依赖 AITER 库：新增自定义 op 依赖 AITER 的 flash_attn_varlen_fp8_pertensor_func，若 AITER 版本更新或函数签名变更可能导致兼容性问题（vllm/_aiter_ops.py）。
  2. 平台限制：FP8 路径仅在 ROCm 和 MI300/MI350 架构上可用，其他 AMD GPU 或 CUDA 环境会报错，需确保配置正确（mm_encoder_attention.py_init_fp8_state）。
  3. 控制流风险：_init_fp8_state 的控制流被修改为优先分支到 AITER，可能影响现有 FlashInfer 路径的初始化，但已验证 CUDA 端行为不变。
  4. 后端枚举同步：引入的新后端 ROCM_AITER_FA 需要 backends.registry 和 selector 同步支持，当前 get_vit_attn_backend 似乎未改动，需确认 selector 是否正确识别。 - 影响：用户：AMD MI300/MI350 用户可通过设置 mm_encoder_attn_dtype='fp8' 与 ROCM_AITER_FA 后端获得 FP8 加速，最高 1.38x 速度提升，多模态任务受益。系统：新增自定义 op 并注册，但经过 torch.compile(fullgraph=True) 验证，不破坏现有图编译。团队：需要维护 AITER 集成，但代码路径与 NVIDIA 路径隔离，不增加 CUDA 维护负担；后续可复用该模式添加更多 AMD 算子。
- 风险标记：依赖 AITER 库，仅 ROCm+MI300/MI350，FP8 初始化控制流变更

关联脉络

- 暂无明显关联 PR