PR #42774 完整报告

vllm-project/vllm

[Perf] Padded nvfp4 quant kernel to remove additional copy, 2.4%~5.7% e2e performance improvement

合并时间：2026-05-19 07:38

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/42774

执行摘要

- 一句话：padded nvfp4 量化 kernel 消除额外 copy
- 推荐动作：值得精读。该 PR 展示了如何通过将后处理步骤融合到 CUDA kernel 中来消除冗余内存访问，是典型的性能优化案例。对于从事量化推理、CUDA kernel 优化或 CUTLASS 集成工作的开发者具有很好的参考价值。建议关注其设计权衡：何时适合将 padding 内联到 kernel，以及如何保持向后兼容。

功能与动机

NVFP4 量化后的输出需要对齐到 CUTLASS 要求的大小，原有的做法是在量化 kernel 之后再执行一次 padding 复制，这带来了额外的带宽开销。通过将 padding 合并到量化 kernel 内部，可以减少一次全局内存访问，提升量化阶段性能，进而提升端到端推理吞吐。

实现拆解

1. 在 vllm/_custom_ops.py 中，修改 create_fp4_output_tensors 和 scaled_fp4_quant 函数，新增可选的 padded_n 参数。当提供 padded_n 时，分配的 output 和 scale buffer 尺寸对应 padded_n（更大的 K 维），允许 CUDA kernel 直接写入 padded 区域。
2. 修改 CUDA kernel 文件 csrc/libtorch_stable/quantization/fp4/nvfp4_quant_kernels.cu 中的 cvt_fp16_to_fp4 和 cvt_fp16_to_fp4_sf_major。引入 outputCols 和 num_padded_cols 参数，调整线程网格计算，仅当 valid_output 时才写入有效数据，padding 区域保持为零。
3. 更新线性层 kernel 的 apply_weights 方法：在 vllm/model_executor/kernels/linear/nvfp4/cutlass.py 和 flashinfer.py 中，移除调用 pad_nvfp4_activation_for_cutlass 的步骤，改为将 padded_n 参数传递给 scaled_fp4_quant，由量化 kernel 直接完成 padding。同时移除不再需要的 pad_nvfp4_activation_for_cutlass 导入。
4. 在测试文件 tests/kernels/quantization/test_nvfp4_quant.py 中，新增 test_quantize_to_fp4_with_padded_output 测试用例，覆盖多种 shape 和 layout，验证 padding 区域为零且有效值与 reference 一致。同时提取了 round_up 工具函数。
5. （附加）删除了独立的 benchmark 脚本 benchmarks/kernels/benchmark_nvfp4_padded_quant.py，避免重复（应 LopezCastroRoberto 审查意见）。

关键文件：

- vllm/_custom_ops.py（模块 量化接口；类别 source；类型 core-logic；符号 create_fp4_output_tensors, scaled_fp4_quant）：核心入口，新增 padded_n 参数控制量化输出 buffer 大小，消除后续 padding 复制
- tests/kernels/quantization/test_nvfp4_quant.py（模块 量化测试；类别 test；类型 test-coverage；符号 round_up, test_quantize_to_fp4_with_padded_output）：新增 paded 输出测试用例，覆盖多种 shape 和 layout，验证 padding 为零及有效值正确
- vllm/model_executor/kernels/linear/nvfp4/cutlass.py（模块 线性层；类别 source；类型 data-contract）：CUTLASS 线性层 kernel 使用 padded_n 替代外部 padding 调用，移除冗余导入
- vllm/model_executor/kernels/linear/nvfp4/flashinfer.py（模块 线性层；类别 source；类型 data-contract）：FlashInfer 线性层 kernel 使用 padded_n 替代外部 padding 调用，移除冗余导入
- csrc/libtorch_stable/quantization/fp4/nvfp4_quant_kernels.cu（模块 CUDA 内核；类别 other；类型 core-logic）：CUDA kernel 修改，引入 outputCols 区分输入 / 输出维度，使 kernel 支持直接写入 padded buffer

关键符号：create_fp4_output_tensors, scaled_fp4_quant, cvt_fp16_to_fp4, cvt_fp16_to_fp4_sf_major, FlashInferCutlassNvFp4LinearKernel.apply_weights, FlashInferTrtllmNvFp4LinearKernel.apply_weights, CutlassNvFp4LinearKernel.apply_weights

关键源码片段

vllm/_custom_ops.py

核心入口，新增 padded_n 参数控制量化输出 buffer 大小，消除后续 padding 复制

```
# vllm/_custom_ops.py — 关键片段
# 修改 create_fp4_output_tensors 以支持 padded_n 参数

def create_fp4_output_tensors(
    m: int,
    n: int,
    device: torch.device,
    is_sf_swizzled_layout: bool,
    padded_n: int | None = None, # <-- 新增：可选的目标 K 维度（含 padding）
) -> tuple[torch.Tensor, torch.Tensor]:
    """
    ...
    When ``padded_n`` is provided, allocate a larger packed-FP4 output/scale
    buffer so the quantization kernel can write CUTLASS-compatible K padding directly
    """
    # 实际分配时使用 padded_n（若提供）否则用原始的 n
    physical_n = padded_n if padded_n is not None else n
    output = torch.empty((m, physical_n // 2), device=device, dtype=torch.uint8)
    output_scale = create_fp4_scale_tensor(m, physical_n, device, is_sf_swizzled_layout)
    return output, output_scale
```

```
# scaled_fp4_quant 同样新增 padded_n 参数


def scaled_fp4_quant(
    input: torch.Tensor,
    input_global_scale: torch.Tensor,
    is_sf_swizzled_layout: bool = True,
    backend: str = "none",
    padded_n: int | None = None, # <-- 新增
) -> tuple[torch.Tensor, torch.Tensor]:
    # ... 断言 ...
    if padded_n is not None:
        assert padded_n >= n
        assert padded_n % block_size == 0
    # 对于 8x4 SF layout 不支持 padded_n，抛出异常（TBD 后续扩展）
    if use_8x4_sf_layout and padded_n is not None and padded_n != n:
        raise ValueError("padded_n is not supported with TRTLLM 8x4 scale layout.")
    if use_8x4_sf_layout:
        # ... 使用 flashinfer 量化路径，不支持 padding
        pass
    else:
        # 预分配包含 padding 的 buffer，调用 .out 变体
        output, output_scale = create_fp4_output_tensors(
            m, n, input.device, is_sf_swizzled_layout, padded_n=padded_n,
        )
        torch.ops._C.scaled_fp4_quant.out(
            input, input_global_scale, is_sf_swizzled_layout,
            output=output, output_scale=output_scale
        )
```

tests/kernels/quantization/test_nvfp4_quant.py

新增 paded 输出测试用例，覆盖多种 shape 和 layout，验证 padding 为零及有效值正确

```
# tests/kernels/quantization/test_nvfp4_quant.py — 新增测试片段

# 全局 round_up 辅助函数（提取为独立函数）
def round_up(x: int, y: int) -> int:
    return (x + y - 1) // y * y

# PADDED_OUTPUT_SHAPES 列表包含非对齐 shape（如 (64, 7152)）
PADDED_OUTPUT_SHAPES = [(128, 48), (128, 80), (150, 48), (150, 80), (64, 7152)]


@pytest.mark.parametrize("shape", PADDED_OUTPUT_SHAPES)
@pytest.mark.parametrize("is_sf_swizzled_layout", [True, False])
@torch.inference_mode()
def test_quantize_to_fp4_with_padded_output(
    shape: tuple[int, int],
    is_sf_swizzled_layout: bool,
```

```
) -> None:
    from vllm._custom_ops import create_fp4_output_tensors

    dtype = torch.float16
    set_random_seed(42)
    torch.set_default_device("cuda:0")
    m, n = shape
    # 将 n 向上取整到 32 的倍数作为 padded_n
    padded_n = round_up(n, 32)
    assert padded_n > n

    x = torch.randn((m, n), dtype=dtype)
    tensor_amax = torch.abs(x).max().to(torch.float32)
    global_scale = FLOAT8_E4M3_MAX * FLOAT4_E2M1_MAX / tensor_amax

    # 计算参考输出（无 padding）
    out_ref, scale_ref = ref_nvfp4_quant(x, global_scale)

    # 使用 padded_n 调用 ops.scaled_fp4_quant
    out, out_scale = ops.scaled_fp4_quant(
        x,
        global_scale,
        is_sf_swizzled_layout=is_sf_swizzled_layout,
        padded_n=padded_n,
    )
    # 验证 output shape 为 (m, padded_n // 2)
    assert out.shape == (m, padded_n // 2)
    # 验证 padding 区域（后半部分量 packed 后）全为零
    assert torch.count_nonzero(out[:, n // 2 :]) == 0

    # 解量化有效部分并与 reference 对比
    out_ans = cast_from_fp4(out[:, : n // 2], m, n)
    torch.testing.assert_close(out_ans, out_ref)

    # 验证 scale 的有效部分
    if is_sf_swizzled_layout:
        scale_ans = recover_swizzled_scales(out_scale, m, padded_n)
        torch.testing.assert_close(scale_ans[:, : n // BLOCK_SIZE], scale_ref)
        assert torch.count_nonzero(scale_ans[:, n // BLOCK_SIZE :]) == 0
    else:
        scale_ans = out_scale.to(torch.float32)
        torch.testing.assert_close(scale_ans[:, : n // BLOCK_SIZE], scale_ref)
        assert torch.count_nonzero(scale_ans[:, n // BLOCK_SIZE :]) == 0
```

评论区精华

审阅中主要围绕性能增益和代码质量。LopezCastroRoberto 质疑 e2e 提升是否足够显著，作者提供 benchmark 数据后确认有效并 approve。关于 sf_major kernel 中变量命名，gemini-code-assist 建议重命名 valid 为 valid_input 以与 swizzled kernel 保持一致，但未在

本 PR 中采纳。另指出 8x4 SF layout 下不支持 padded_n，作者添加 TODO 记录。独立 benchmark 脚本被要求移除，作者已执行。

- *e2e* 性能提升质疑 (performance): LCR 确认提升有效并 approve
- 独立 benchmark 脚本必要性 (other): 已移除独立 benchmark 脚本
- sf_major kernel 变量命名一致性 (style): 未采纳，PR 合入时仍使用原命名
- 8x4 SF layout 不支持 padded_n (design): 添加了 TODO 注释，计划未来扩展支持

风险与影响

- 风险：主要风险包括：(1) CUDA kernel 修改可能引入精度或数值问题，但已通过新增 padding 测试覆盖多种非对齐形状；(2) TRTLLM 8x4 SF layout 与 padded_n 冲突，scaled_fp4_quant 中已添加断言和 TODO 拦截；(3) 移除 pad_nvfp4_activation_for_cutlass 后，若某些自定义后端未设置 weights_padding_cols，padded_n 计算可能异常，但默认 padded_n=None 维持向后兼容；(4) 性能风险低，benchmark 已验证正向收益。
- 影响：直接影响使用 NVFP4 量化（modelopt_fp4）且在 sm_100+ Nvidia GPU 上运行的模型推理性能，特别是采用 CUTLASS/FlashInfer 后端的线性层。端到端吞吐提升 2.4%~5.7%，kernel 级提升 15%~40%。对 TRTLLM 8x4 layout 用户无影响（会触发 ValueError）。对不使用 NVFP4 的模型无影响。修改集中在量化路径，不涉及调度器、前端等模块。
- 风险标记：CUDA kernel 修改，量化精度风险，兼容性：TRTLLM 8x4 layout 不支持 padded_n

关联脉络

- 暂无明显关联 PR