PR #41882 完整报告

vllm-project/vllm

Add NVFP4 all-gather GEMM fusion for AsyncTP

合并时间：2026-05-10 09:13

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/41882

执行摘要

- 一句话：为 AsyncTP 添加 NVFP4 all-gather GEMM 融合路径
- 推荐动作：推荐精读，尤其关注 collective_fusion.py 中 FlashInferAllGatherFP4Pattern 的 pattern 与 replacement 设计，以及 sequence_parallelism.py 中 NVFP4 量化与序列平行的整合方式。对推理性能优化感兴趣的同学可以关注 reduce-scatter 融合的后续进展。

功能与动机

PR 描述指出，该 PR 将 NVFP4 FlashInfer all-gather + GEMM 路径接入 AsyncTP，以覆盖 SP + AsyncTP 在 NVFP4 下的性能收益。由于 PyTorch 缺少 NVFP4 感知的 fused GEMM + reduce-scatter 支持，当前仅实现 all-gather 融合。

实现拆解

1. 在 vllm/utils/flashinfer.py 中添加 flashinfer_scaled_fp4_mm_out 函数，封装了 FlashInfer mm_fp4_ 的调用，支持 pre-allocated output buffer。
2. 在 collective_fusion.py 中添加 NVFP4 专用的 MM 适配器 _flashinfer_fp4_mm_out，以及 fused all-gather + GEMM 操作的真实实现和 fake 实现，支持对称内存式流水线 all-gather。
3. 定义 FlashInferAllGatherFP4Pattern 类，通过 pattern 和 replacement 匹配 QKV/MLP 中的 all-gather + MM + FP4 量化组合，注册到 FusionPass。
4. 在 sequence_parallelism.py 中添加 FirstAllReduceRMSNormStaticNVFP4Pattern 和 MiddleAllReduceRMSNormStaticNVFP4Pattern，支持 NVFP4 量化在 SP 中的 AllReduce→RMSNorm→Quant 模式融合。
5. 添加三组测试：
   - test_tp2_async_tp_nvfp4_fusions：验证融合计数。
   - test_async_tp_pass_nvfp4_correctness：正确性对比。
   - test_tp_sp_nvfp4_generation：SP 模式生成测试。

关键文件：

- vllm/compilation/passes/fusion/collective_fusion.py（模块 编译融合；类别 source；类型 core-logic；符号 _flashinfer_fp4_mm_out, fused_all_gather_flashinfer_fp4_matmul_fake, fused_all_gather_flashinfer_fp4_matmul, fp4_shard_consumer）：核心变更文件，

新增 NVFP4 all-gather + GEMM 融合的自定义操作和模式注册。

- vllm/compilation/passes/fusion/sequence_parallelism.py（模块 编译融合；类别 source；类型 core-logic；符号 FirstAllReduceRMSNormStaticNVFP4Pattern, MiddleAllReduceRMSNormStaticNVFP4Pattern）：添加 NVFP4 序列并行模式，将 AllReduce→RMSNorm→Quant 转换为 ReduceScatter→RMSNorm→Quant→AllGather。
- vllm/utils/flashinfer.py（模块 FlashInfer 工具；类别 source；类型 core-logic；符号 flashinfer_scaled_fp4_mm_out）：新增 flashinfer_scaled_fp4_mm_out 函数，封装 FlashInfer FP4 mm 的 out 变体调用。
- tests/compile/correctness_e2e/test_async_tp.py（模块 集成测试；类别 test；类型 test-coverage；符号 test_async_tp_pass_nvfp4_correctness）：新增 test_async_tp_pass_nvfp4_correctness 测试用例，验证 NVFP4 AsyncTP 正确性。
- tests/compile/fusions_e2e/test_tp2_async_tp.py（模块 集成测试；类别 test；类型 test-coverage；符号 test_tp2_async_tp_nvfp4_fusions）：新增 test_tp2_async_tp_nvfp4_fusions 测试用例，验证融合模式触发次数。
- tests/compile/correctness_e2e/test_sequence_parallel.py（模块 集成测试；类别 test；类型 test-coverage；符号 test_tp_sp_nvfp4_generation）：新增 test_tp_sp_nvfp4_generation 测试用例，覆盖 Sequence Parallelism 下的 NVFP4 生成。
- tests/compile/fullgraph/test_toy_llama.py（模块 单测；类别 test；类型 test-coverage）：调整以适配 NVFP4 后端的已知行为变化。

关键符号：_flashinfer_fp4_mm_out, fused_all_gather_flashinfer_fp4_matmul, fused_all_gather_flashinfer_fp4_matmul_fake, FlashInferAllGatherFP4Pattern, FirstAllReduceRMSNormStaticNVFP4Pattern, MiddleAllReduceRMSNormStaticNVFP4Pattern, flashinfer_scaled_fp4_mm_out, test_async_tp_pass_nvfp4_correctness, test_tp2_async_tp_nvfp4_fusions, test_tp_sp_nvfp4_generation

关键源码片段

vllm/compilation/passes/fusion/collective_fusion.py

核心变更文件，新增 NVFP4 all-gather + GEMM 融合的自定义操作和模式注册。

```
# vllm/compilation/passes/fusion/collective_fusion.py

def fused_all_gather_flashinfer_fp4_matmul(
    A_shard: torch.Tensor,
    B: torch.Tensor,
    A_scale_shard: torch.Tensor,
    B_scale: torch.Tensor,
    alpha: torch.Tensor,
    gather_dim: int,
    group_name: str,
    out_dtype: torch.dtype | None = None,
    view_a_scale_as_fp8: bool = False,
    use_8x4_sf_layout: bool = False,
```

```
    backend: str = "cutlass",
) -> torch.Tensor:
    # 只支持 gather_dim=0（按行拼接）
    assert gather_dim == 0, "FP4 symm_mem adapter only supports gather_dim=0"
    assert A_shard.ndim == 2 and A_scale_shard.ndim == 2 and B.ndim == 2

    # 可选：将 scale 重解释为 FP8 以复用接口
    if view_a_scale_as_fp8:
        A_scale_shard = A_scale_shard.view(torch.float8_e4m3fn)

    group = c10d._resolve_process_group(group_name)
    world_size = group.size()

    # 预分配完整输出张量
    output = A_shard.new_empty(
        A_shard.shape[0] * world_size,
        B.shape[1],
        dtype=out_dtype or torch.bfloat16,
    )
    output_shards = output.chunk(world_size)

    # 分配 all-gather 目标缓冲区（非对称内存，可能带来拷贝开销）
    A = A_shard.new_empty(A_shard.shape[0] * world_size, A_shard.shape[1])
    A_scale = A_scale_shard.new_empty(
        A_scale_shard.shape[0] * world_size,
        A_scale_shard.shape[1],
    )

    # 流水线 all-gather：每个 rank 数据到达后立即计算局部 GEMM
    def fp4_shard_consumer(shards: list[torch.Tensor], rank: int) -> None:
        _flashinfer_fp4_mm_out(
            shards[0],
            B,
            scale_a=shards[1],
            scale_b=B_scale,
            alpha=alpha,
            out=output_shards[rank],
            out_dtype=out_dtype,
            use_8x4_sf_layout=use_8x4_sf_layout,
            backend=backend,
        )

    # 融合 all-gather 与计算
    torch.distributed._symmetric_memory._pipelined_multi_all_gather(
        [A, A_scale],
        [A_shard, A_scale_shard],
        group_name,
        stream_consumer=fp4_shard_consumer,
    )
```

```
    return output
```

vllm/compilation/passes/fusion/sequence_parallelism.py

添加 NVFP4 序列并行模式，将 AllReduce→RMSNorm→Quant 转换为 ReduceScatter→RMSNorm→Quant→AllGather。

```
# vllm/compilation/passes/fusion/sequence_parallelism.py

class FirstAllReduceRMSNormStaticNVFP4Pattern(_SequenceParallelPatternHelper):
    def get_inputs(self) -> list[torch.Tensor]:
        # 创建示例张量供模式匹配器使用
        input = self.empty([8, 16])
        weight = self.empty([16])
        input_global_scale = self.empty_f32([1, 1])
        quant_output = torch.empty([8, 8], device=self.device, dtype=torch.uint8)
        output_scale = torch.empty([128, 4], device=self.device, dtype=torch.int32)
        return [input, weight, input_global_scale, quant_output, output_scale]

    def register(self, pm_pass: PatternMatcherPass) -> None:
        def pattern(
            input, weight, input_global_scale, quant_output, output_scale
        ) -> tuple[torch.Tensor, torch.Tensor, torch.Tensor]:
            # 原始图：AllReduce -> RMSNorm -> NVFP4 Quant
            all_reduce = self._all_reduce(input)
            rms = vllm.ir.ops.rms_norm(all_reduce, weight, self.epsilon)
            quant = auto_functionalized(
                SCALED_FP4_QUANT_OUT_OVERLOAD,
                input=rms,
                input_scale=input_global_scale,
                is_sf_swizzled_layout=True,
                output=quant_output,
                output_scale=output_scale,
            )
            return quant[1], all_reduce, quant[2]

        def replacement(
            input, weight, input_global_scale, quant_output, output_scale
        ) -> tuple[torch.Tensor, torch.Tensor, torch.Tensor]:
            # 替换图：ReduceScatter -> RMSNorm -> NVFP4 Quant -> AllGather
            reduce_scatter = self._reduce_scatter(input)
            rms = vllm.ir.ops.rms_norm(reduce_scatter, weight, self.epsilon)
            rms = torch.ops.aten.view.default(rms, [-1, rms.shape[-1]])
            quant = SCALED_FP4_QUANT_DEFAULT_OVERLOAD(
                rms, input_global_scale, True,
            )
            return (
                self._all_gather(quant[0]),
                reduce_scatter,
                self._all_gather(quant[1]),
```

```
            )

        pm.register_replacement(
            pattern, replacement, self.get_inputs(), pm.fwd_only, pm_pass
        )
```

vllm/utils/flashinfer.py

新增 flashinfer_scaled_fp4_mm_out 函数，封装 FlashInfer FP4 mm 的 out 变体调用。

```
# vllm/utils/flashinfer.py

def flashinfer_scaled_fp4_mm_out(
    a: torch.Tensor,
    b: torch.Tensor,
    block_scale_a: torch.Tensor,
    block_scale_b: torch.Tensor,
    alpha: torch.Tensor,
    out: torch.Tensor,
    out_dtype: torch.dtype | None,
    use_8x4_sf_layout: bool,
    backend: str,
) -> torch.Tensor:
    # 所有张量必须是 2D，并且 out 已经预分配好
    assert a.ndim == 2 and b.ndim == 2 and out.ndim == 2
    assert block_scale_a.ndim == 2 and block_scale_b.ndim == 2
    assert a.shape[1] == b.shape[0]
    assert out.shape == (a.shape[0], b.shape[1])

    # 对 cutlass / cudnn 后端，将 scale 重解释为 uint8
    if backend in ("cutlass", "cudnn"):
        if block_scale_a.dtype != torch.uint8:
            block_scale_a = block_scale_a.view(torch.uint8)
        if block_scale_b.dtype != torch.uint8:
            block_scale_b = block_scale_b.view(torch.uint8)

    from flashinfer import mm_fp4 as flashinfer_mm_fp4_
    # 调用 FlashInfer 的 FP4 matmul，输出写入预先分配的 out
    flashinfer_mm_fp4_(
        a, b, block_scale_a, block_scale_b, alpha,
        out_dtype or out.dtype,
        out=out,
        block_size=16,
        use_8x4_sf_layout=use_8x4_sf_layout,
        backend=backend,
    )
    return out
```

评论区精华

- 对称内存建议：gemini-code-assist[bot] 建议对中间缓冲区使用对称内存以避免不必要的拷贝，作者未公开回应但当前实现仍使用 new_empty。
- Double view 逻辑简化：bot 指出 float8_uint8 双重 view 可简化，建议直接 view 为 uint8。后续 commit 中变量命名已调整（从 STATIC_FP4_QUANT_OP 改为 SCALED_FP4_QUANT_OUT_OVERLOAD），但 double view 是否完全清理需确认。
- Reduce-scatter 可行性：ProExpertProg 提问 reduce-scatter 是否也是 trivially 可支持的，自答输入已经是列并行，无需额外 scale 通信，但当前保持 disable，留作后续。
- CI 测试覆盖：ProExpertProg 要求将新测试加入正确性 CI，PR 已包含测试但需要确认是否纳入了 CI 流水线。
  - 使用对称内存优化中间缓冲区 (performance): 未在 PR 中明显采纳，当前仍使用 new_empty。
  - 简化 a_scale_view 的 double view 逻辑 (style): 后续可能已调整（从 STATIC_FP4_QUANT_OP 重命名），但 double view 是否完全消除需确认。
  - NVFP4 reduce-scatter 融合是否必要 (design): 确认设计有意，留作后续。
  - 将 NVFP4 测试加入 CI (testing): 测试已添加，但需要确保 CI 配置包含这些测试。
  - 量化操作符命名 (style): 后续提交中已改为 SCALED_FP4_QUANT_OUT_OVERLOAD。

风险与影响

- 风险：
  - 量化类型检查：NVFP4 相关代码仅在 torch.ops._C.scaled_fp4_quant 存在时注册，若 FlashInfer 不可用或设备不支持则跳过，不会影响现有 FP8 路径。
  - 性能风险：中间缓冲区使用 new_empty 而非对称内存，可能带来额外拷贝开销，削弱 AsyncTP 收益（尤其小 batch 场景）。
  - 硬件依赖：仅 Blackwell（SM100）和 FlashInfer 支持，其他平台无影响。
  - 测试覆盖：目前仅测试 TP=2，缺少更大规模 TP 或不同序列长度组合的测试。
- 影响：
  - 用户：使用 NVFP4 量化的 Llama 等模型在启用 SP + AsyncTP 时获得 0.89%-13.54% 的吞吐提升，无需修改模型代码。
  - 系统：新增的 fused 操作会增加编译通道的模式匹配负担，但仅在 NVFP4 路径下生效。
  - 团队：降低了后续添加 reduce-scatter 融合的门槛，为其他量化类型提供了参考模式。
  - 风险标记：仅 Blackwell 支持，依赖 FlashInfer，对称内存未使用，Reduce-scatter 未融合，仅 2 GPU 测试

关联脉络

- PR #41769 [Quantization] Add ModelOpt NVFP4 W4A16 (4-bit weights, fp16/bf16 activations) support: 提供了 NVFP4 量化基础，本 PR 在此基础上增加融合通信支持。