PR #40408 完整报告

vllm-project/vllm

[Perf] Batch invariance with Cutlass fp8 support, 28.9% E2E latency improvement

合并时间：2026-05-12 00:20

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/40408

执行摘要

- 一句话：使用 Cutlass FP8 实现批量不变性，延迟降低 28.9%
- 推荐动作：该 PR 值得精读，尤其关注：1）如何通过固定 CUTLASS 配置实现 batch invariance 并保持正确性；2）FP8 线性层 apply 的分支设计兼顾性能与回退。对使用 FP8 批处理推理的团队有直接影响。

功能与动机

避免 FP8 量化 / 反量化开销，提升 batch invariant 模式下的推理性能。PR body 明确指出“Use cutlass fp8 to avoid quantize/dequantize overhead”。

实现拆解

1. 新增 CUTLASS batch invariant 内核：在 csrc/libtorch_stable/quantization/w8a8/cutlass/c3x/ 下的 scaled_mm_sm{90,100,120}_fp8_dispatch.cuh 及 scaled_mm_c2x_sm89_fp8_dispatch.cuh 中新增 cutlass_gemm_*_batch_invariant_dispatch 和 cutlass_scaled_mm_*_batch_invariant_epilogue 函数。这些函数使用固定的 CUTLASS 配置（M=64），不随实际 M 变化，从而保证输出与 batch size 无关。
2. 修改 Python 量化层：在 vllm/model_executor/layers/quantization/fp8.py 和 online/fp8.py 的 apply 方法中添加分支——当 VLLM_BATCH_INVARIANT 启用且 self.fp8_linear 是 CutlassFP8ScaledMMLinearKernel 实例时，直接调用 self.fp8_linear.apply_weights(layer, x, bias)，跳过原有的 BF16 反量化路径。注释也相应更新。
3. 更新 .cu 文件绑定：在 scaled_mm_sm{90,100,120}_fp8.cu 和 scaled_mm_c2x.cu 中导入并调用新的 batch_invariant_epilogue 函数，使其对 Python 可见。
4. 新增测试：添加 tests/v1/determinism/test_cutlass_batch_invariance.py，使用 TestFP8Layer 强制 CutlassFP8ScaledMMLinearKernel，验证在不同 batch size 和 weight shape 下，特定 token 的输出严格与 batch size 无关（assert_close 要求 rtol=0, atol=0）。

关键文件：

- tests/v1/determinism/test_cutlass_batch_invariance.py（模块 批量测试；类别 test；类型 test-coverage；符号 setup_cuda, test_cutlass_fp8_batch_invariant_fixed_config）：

新增测试文件，验证 Cutlass FP8 内核在不同 batch size 和 weight shape 下的 batchinvariance 性质，确保功能正确性。

- vllm/model_executor/layers/quantization/fp8.py（模块 量化层；类别 source；类型 data-contract）：核心量化层文件，修改了 apply 方法以支持 Cutlass FP8 batch invariant 路径，并更新了导入。
- vllm/model_executor/layers/quantization/online/fp8.py（模块 量化层；类别 source；类型 data-contract）：在线 FP8 量化层，与 fp8.py 类似修改 apply 方法以支持 Cutlass FP8 batch invariant 路径。
- csrc/libtorch_stable/quantization/w8a8/cutlass/c3x/scaled_mm_sm90_fp8_dispatch.cuh（模块 CUTLASS 内核；类别 other；类型 core-logic）：新增 SM90 专用的 batch invariant dispatch 和 epilogue 函数，是 CUTLASS 内核的核心变更。
- csrc/libtorch_stable/quantization/w8a8/cutlass/c3x/scaled_mm_sm100_fp8_dispatch.cuh（模块 CUTLASS 内核；类别 other；类型 core-logic）：SM100 版本 batch invariant dispatch，与 SM90 类似，新增对应函数。
- csrc/libtorch_stable/quantization/w8a8/cutlass/c3x/scaled_mm_sm120_fp8_dispatch.cuh（模块 CUTLASS 内核；类别 other；类型 core-logic）：SM120 版本 batch invariant dispatch，新增对应函数。
- csrc/libtorch_stable/quantization/w8a8/cutlass/scaled_mm_c2x_sm89_fp8_dispatch.cuh（模块 CUTLASS 内核；类别 other；类型 core-logic）：SM89 (Ada) 版本的 batch invariant dispatch，保持所有架构一致性。
- csrc/libtorch_stable/quantization/w8a8/cutlass/c3x/scaled_mm_sm120_fp8.cu（模块 CUTLASS 绑定；类别 other；类型 dependency-wiring）：CUDA 源文件，绑定 SM120 的 batch_invariant_epilogue 到 Python，是连接 C++ 和 Python 的关键。
- csrc/libtorch_stable/quantization/w8a8/cutlass/scaled_mm_c2x.cu（模块 CUTLASS 绑定；类别 other；类型 dependency-wiring）：CUDA 源文件，绑定 SM89 的 batch_invariant_epilogue。
- csrc/libtorch_stable/quantization/w8a8/cutlass/c3x/scaled_mm_sm100_fp8.cu（模块 CUTLASS 绑定；类别 other；类型 dependency-wiring）：CUDA 源文件，绑定 SM100 的 batch_invariant_epilogue。
- csrc/libtorch_stable/quantization/w8a8/cutlass/c3x/scaled_mm_sm90_fp8.cu（模块 CUTLASS 绑定；类别 other；类型 dependency-wiring）：CUDA 源文件，绑定 SM90 的 batch_invariant_epilogue。

关键符号：Fp8LinearMethod.apply, _Fp8OnlineLinearBase.apply, cutlass_gemm_sm90_fp8_batch_invariant_dispatch, cutlass_scaled_mm_sm90_fp8_batch_invariant_epilogue, test_cutlass_fp8_batch_invariant_fixed_config

关键源码片段

tests/v1/determinism/test_cutlass_batch_invariance.py

新增测试文件，验证 Cutlass FP8 内核在不同 batch size 和 weight shape 下的 batch invariance 性质，确保功能正确性。

```
# tests/v1/determinism/test_cutlass_batch_invariance.py

import pytest
import torch
import vllm.envs as envs
from tests.utils import TestFP8Layer, requires_fp8
from vllm.model_executor.kernels.linear.scaled_mm.cutlass import
CutlassFP8ScaledMMLinearKernel
from vllm.model_executor.layers.quantization.utils.quant_utils import kFp8DynamicTokenSym,
kFp8StaticTensorSym
from vllm.platforms import current_platform

pytest.importorskip("torch.cuda")

@pytest.fixture(autouse=True)
def setup_cuda():
    if not current_platform.is_cuda():
        pytest.skip("CUTLASS FP8 kernels require CUDA.")
    torch.set_default_device("cuda")

@requires_fp8
@pytest.mark.parametrize("weight_shape", [(1024, 2048), (4608, 4096)])
@pytest.mark.parametrize("batch_size", [1, 16, 17, 32, 64, 65, 256, 257])
@torch.inference_mode()
def test_cutlass_fp8_batch_invariant_fixed_config(
    weight_shape: tuple[int, int],
    batch_size: int,
    default_vllm_config,
    monkeypatch: pytest.MonkeyPatch,
):
    # 启用 batch invariant 环境变量
    monkeypatch.setenv("VLLM_BATCH_INVARIANT", "1")
    monkeypatch.setattr(envs, "VLLM_BATCH_INVARIANT", True)

    torch.manual_seed(0)
    # 构造 FP8 层，强制使用 CutlassFP8ScaledMMLinearKernel
    layer = TestFP8Layer(
        weight_shape=weight_shape,
        activation_quant_key=kFp8DynamicTokenSym,
        weight_quant_key=kFp8StaticTensorSym,
        input_dtype=torch.bfloat16,
        out_dtype=torch.bfloat16,
        device=torch.device("cuda"),
        force_kernel=CutlassFP8ScaledMMLinearKernel,
    )
    assert isinstance(layer.kernel, CutlassFP8ScaledMMLinearKernel)
```

```
    in_features = weight_shape[1]
    # 创建一个 needle token，作为检测 batch invariant 的锚点
    needle = torch.randn((1, in_features), device="cuda", dtype=torch.bfloat16)
    baseline = layer(needle)[0] # 单个 token 的输出作为基准

    # 创建 filler 用来组装不同 batch size
    filler = torch.randn(
        (max(batch_size - 1, 0), in_features), device="cuda", dtype=torch.bfloat16
    )

    # 将 needle 放在 batch 的最前面和最后面
    front_batch = torch.cat([needle, filler], dim=0)
    back_batch = torch.cat([filler, needle], dim=0)

    front_output = layer(front_batch)[0]
    back_output = layer(back_batch)[-1]

    # 严格校验：无论 batch size 和 needle 位置，输出与 baseline 一致
    torch.testing.assert_close(front_output, baseline, rtol=0, atol=0)
    torch.testing.assert_close(back_output, baseline, rtol=0, atol=0)
```

vllm/model_executor/layers/quantization/fp8.py

核心量化层文件，修改了 apply 方法以支持 Cutlass FP8 batch invariant 路径，并更新了导入。

```
# vllm/model_executor/layers/quantization/fp8.py (partial)

import torch
import vllm.envs as envs
from vllm.model_executor.kernels.linear.scaled_mm import (
    CutlassFP8ScaledMMLinearKernel,
    MarlinFP8ScaledMMLinearKernel,
)

# ... ( 类定义 )

def apply(
    self,
    layer: torch.nn.Module,
    x: torch.Tensor,
    bias: torch.Tensor | None = None,
) -> torch.Tensor:
    # 当启用 VLLM_BATCH_INVARIANT 时，优先使用直接 FP8 路径
    # 如果底层内核是 CutlassFP8ScaledMMLinearKernel 且非 block 量化，
    # 则直接调用 apply_weights，避免 BF16 反量化开销
    if envs.VLLM_BATCH_INVARIANT:
        if self.block_quant:
            assert self.weight_block_size is not None
            return self.fp8_linear.apply_weights(layer, x, bias)
```

```
        else:
            # 新分支：直接使用 Cutlass FP8 计算
            if isinstance(self.fp8_linear, CutlassFP8ScaledMMLinearKernel):
                return self.fp8_linear.apply_weights(layer, x, bias)

            # 反量化回 BF16 并执行 GEMM (fallback)
            weight_fp8 = layer.weight.to(torch.bfloat16)
            weight_scale = layer.weight_scale.to(torch.bfloat16)
            if weight_scale.numel() == 1:
                weight_bf16 = weight_fp8 * weight_scale
            else:
                # 多 scale 处理 ( 如 QKV 融合 )
                if weight_scale.dim() == 1 and weight_scale.shape[0] == weight_fp8.shape[0]:
                    weight_bf16 = weight_fp8 * weight_scale.unsqueeze(1)
                else:
                    weight_bf16 = weight_fp8 * weight_scale
            return torch.nn.functional.linear(x, weight_bf16.t(), bias)

    # 非 batch invariant 模式：使用 Marlin 或默认的 FP8 scaled GEMM
    if self.use_marlin:
        return self.fp8_linear.apply_weights(layer, x, bias)
    return self.fp8_linear.apply_weights(layer, x, bias)
```

csrc/libtorch_stable/quantization/w8a8/cutlass/c3x/scaled_mm_sm90_fp8_dispatch.cuh

新增 SM90 专用的 batch invariant dispatch 和 epilogue 函数，是 CUTLASS 内核的核心变更。

```
// csrc/libtorch_stable/quantization/w8a8/cutlass/c3x/scaled_mm_sm90_fp8_dispatch.cuh
// (partial)

template <typename InType, typename OutType, bool EnableBias,
          typename... EpilogueArgs>
inline void cutlass_gemm_sm90_fp8_batch_invariant_dispatch(
    torch::stable::Tensor& out, torch::stable::Tensor const& a,
    torch::stable::Tensor const& b, torch::stable::Tensor const& a_scales,
    torch::stable::Tensor const& b_scales, EpilogueArgs&&... args) {
  // 该 dispatch 使用固定 CUTLASS 配置，不依赖于 M（batch size）
  // 确保 batch invariance：输出与 M 无关
  static_assert(std::is_same<InType, cutlass::float_e4m3_t>());
  // 检查张量类型为 FP8 e4m3
  STD_TORCH_CHECK(a.scalar_type() == torch::headeronly::ScalarType::Float8_e4m3fn);
  STD_TORCH_CHECK(b.scalar_type() == torch::headeronly::ScalarType::Float8_e4m3fn);

  // 根据 N 维大小选择不同的 CUTLASS 配置
  using Cutlass3xGemmM64_N1280 = typename sm90_fp8_config_M64_N1280<InType, OutType,
  EnableBias>::Cutlass3xGemm;
  using Cutlass3xGemmM64_N8192 = typename sm90_fp8_config_M64_N8192<InType, OutType,
  EnableBias>::Cutlass3xGemm;
```

```
  uint32_t const n = b.size(1); // 输出特征维度
  if (n <= 1280) {
    return cutlass_gemm_caller_sm90_fp8<Cutlass3xGemmM64_N1280>(
        out, a, b, b_scales, a_scales, std::forward<EpilogueArgs>(args)...);
  }
  return cutlass_gemm_caller_sm90_fp8<Cutlass3xGemmM64_N8192>(
      out, a, b, b_scales, a_scales, std::forward<EpilogueArgs>(args)...);
}

template <bool EnableBias, typename... EpilogueArgs>
void cutlass_scaled_mm_sm90_fp8_batch_invariant_epilogue(
    torch::stable::Tensor& out, torch::stable::Tensor const& a,
    torch::stable::Tensor const& b, torch::stable::Tensor const& a_scales,
    torch::stable::Tensor const& b_scales, EpilogueArgs&&... epilogue_args) {
  // 检查输入类型为 FP8
  STD_TORCH_CHECK(a.scalar_type() == torch::headeronly::ScalarType::Float8_e4m3fn);
  STD_TORCH_CHECK(b.scalar_type() == torch::headeronly::ScalarType::Float8_e4m3fn);

  // 根据输出数据类型选择 bf16 或 half 的 dispatch
  if (out.scalar_type() == torch::headeronly::ScalarType::BFloat16) {
    return cutlass_gemm_sm90_fp8_batch_invariant_dispatch<cutlass::float_e4m3_t, cutlass::
    bfloat16_t, EnableBias>(
        out, a, b, a_scales, b_scales, std::forward<EpilogueArgs>(epilogue_args)...);
  } else {
    STD_TORCH_CHECK(out.scalar_type() == torch::headeronly::ScalarType::Half);
    return cutlass_gemm_sm90_fp8_batch_invariant_dispatch<cutlass::float_e4m3_t, cutlass::
    half_t, EnableBias>(
        out, a, b, a_scales, b_scales, std::forward<EpilogueArgs>(epilogue_args)...);
  }
}
```

评论区精华

- Batch invariant 属性的质疑：tlrmchlsmth 指出 CutlassFP8ScaledMMLinearKernel 的内核行为可能随未来调优而失去 batch invariance 属性。作者 yewentao256 回应已附上测试确保任意 M 下输出正确，且 CI 会运行这些单元测试。
- 注释表述更新：ElizaWszola 询问注释中从“prefer DeepGEMM”改为“prefer direct FP8”是否意味着 block 版本也优先 direct FP8。作者解释“direct FP8”包含了 DeepGEMM。
- CUTLASS dispatch 注释需求：tlrmchlsmth 建议在 sm100 和 sm120 dispatch 中添加注释，明确说明配置需要独立于 M 的原因（batch invariance）。作者确认已添加。
  - Batch invariant 属性的质疑与确认 (design): yewentao256 回应已附上测试确保任意 M 下正确，且 CI 会运行这些单元测试，属性暂时可靠。
  - 注释中 DeepGEMM 的表述更新 (style): yewentao256 解释 'direct FP8' 包含 DeepGEMM，注释已准确更新。
  - CUTLASS dispatch 注释需求 (documentation): yewentao256 确认已按要求添加注释。

风险与影响

- 风险：
  - 批次无关性假设风险：当前实现假设固定配置（M=64）在所有情况下都保证输出独立于 M，但若未来对 CUTLASS 内核进行算术优化改变计算结果顺序，可能打破该假设。测试覆盖的 shape 有限（仅两种 weight_shape 和六个 batch size），可能遗漏边界情况。
  - 性能风险：固定 M 配置在部分大 batch 场景下可能非最优，但 batch invariance 要求必须固定，此权衡可接受。
  - 维护同步成本：新增的 batch_invariant_dispatch 系列函数与原始 dispatch 重复较多代码，后续主线内核升级时需要同步更新，增加维护负担。
  - 平台覆盖：SM89/90/100/120 均得到支持，但缺少对其他 GPU 架构（如 SM80）的 fallback 测试，若在未支持的架构上误用可能导致运行时错误。
- 影响：
  - 用户影响：启用 VLLM_BATCH_INVARIANT=1 后，使用 FP8 量化且支持 Cutlass 的模型获得显著性能提升（延迟降低约 29%），且输出不再依赖 batch size，简化了批处理调优。若使用 Marlin 或其他量化 kernel 则行为不变。
  - 系统影响：新增约 300 行 C++ 和 Python 代码，主要影响量化层和 CUTLASS 内核调度。编译时间略有增加。
  - 团队影响：需要维护 batch invariant 内核配置与主内核同步，确保两者行为一致。
  - 风险标记：批次无关性假设，CUTLASS 配置覆盖，测试 shape 有限

关联脉络

- PR #41993 [Refactor] Cleanup batch invariant dead code: 本 PR 为 batch invariant 模式添加 Cutlass FP8 支持，与之前清理 batch invariant 代码（#41993）属于同一功能演进线。