PR #42139 完整报告

vllm-project/vllm

[XPU][MoE] support block_fp8_moe on xpu

合并时间：2026-06-05 08:36

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/42139

执行摘要

- 一句话：XPU 支持 Block FP8 MoE 量化
- 推荐动作：该 PR 值得快速合入，变更简洁且聚焦。建议关注后续是否添加针对 Block FP8 的专项测试以覆盖更多量化组合。设计上继承现有架构，可读性强。

功能与动机

需要支持 Block FP8 量化的 MoE 模型（如 Qwen/Qwen3-30B-A3B-Instruct-2507-FP8）在 XPU 设备上的推理。PR body 中贴出了测试通过截图。

实现拆解

1. 在 xpu_moe.py 中新增 XPUExpertsBlockFp8 子类：继承自 XPUExperts，在 __init__ 中将 self.is_block_fp8 设为 True，并实现 _supports_quant_scheme 静态方法声明支持的量化组合 (kFp8Static128BlockSym, kFp8Dynamic128Sym)。同时在基类 XPUExperts.__init__ 中新增 self.is_block_fp8 = False 以初始化该属性，并在 apply 方法中将 is_block_fp8 传递给底层 XpuFusedMoe 内核。
2. 更新 fp8.py oracle 中的后端映射：在 backend_to_kernel_cls 的 Fp8MoeBackend.XPU 分支中导入并返回 XPUExpertsBlockFp8，使得 Block FP8 量化方案能被路由到正确的专家实现类。
3. 更新 CI 配置文件 test-intel.yaml：在 XPU example test 命令中添加了以 VLLM_XPU_FUSED_MOE_USE_REF=1 环境变量运行 Qwen3-30B-A3B-Instruct-2507-FP8 模型的测试用例，用于验证 Block FP8 MoE 功能。

关键文件：

- vllm/model_executor/layers/fused_moe/experts/xpu_moe.py（模块 MoE 专家；类别 source；类型 core-logic；符号 XPUExpertsBlockFp8, init, _supports_quant_scheme）：核心变更文件：新增 XPUExpertsBlockFp8 类，继承 XPUExperts 并设置 is_block_fp8 = True；基类 __init__ 和 apply 方法同步调整以支持新属性。
- vllm/model_executor/layers/fused_moe/oracle/fp8.py（模块 MoE 路由；类别 source；类型 data-contract）：修改 oracle 以在 XPU 后端返回新增的 XPUExpertsBlockFp8 类，确保 Block FP8 量化方案能被正确路由。
- .buildkite/intel_jobs/test-intel.yaml（模块 CI 配置；类别 config；类型 configuration）：添加 Block FP8 MoE 模型的 CI 测试用例，使用 VLLM_XPU_FUSED_MOE_USE_REF=1

环境变量运行 Qwen3-30B-A3B-Instruct-2507-FP8 模型以验证功能。

关键符号：XPUExperts.init, XPUExperts.apply, XPUExpertsBlockFp8.init, XPUExpertsBlockFp8._supports_quant_scheme

关键源码片段

vllm/model_executor/layers/fused_moe/experts/xpu_moe.py

核心变更文件：新增 XPUExpertsBlockFp8 类，继承 XPUExperts 并设置 is_block_fp8 = True；基类 __init__ 和 apply 方法同步调整以支持新属性。

```
# SPDX-License-Identifier: Apache-2.0
# SPDX-FileCopyrightText: Copyright contributors to the vLLM project
import torch

import vllm.model_executor.layers.fused_moe.modular_kernel as mk
from vllm.model_executor.layers.fused_moe.activation import MoEActivation
from vllm.model_executor.layers.fused_moe.config import (
    FusedMoEConfig,
    FusedMoEParallelConfig,
    FusedMoEQuantConfig,
)
from vllm.model_executor.layers.fused_moe.topk_weight_and_reduce import (
    TopKWeightAndReduceNoOP,
)
from vllm.model_executor.layers.quantization.utils.quant_utils import (
    QuantKey,
    kFp8Dynamic128Sym,
    kFp8DynamicTensorSym,
    kFp8Static128BlockSym, # Block FP8 静态权重量化 key
    kFp8StaticTensorSym,
    kInt4Static,
    kMxfp4Static,
    kMxfp8Dynamic,
    kMxfp8Static,
)
from vllm.platforms import current_platform

if current_platform.is_xpu():
    from vllm_xpu_kernels.fused_moe_interface import XpuFusedMoe

class XPUExperts(mk.FusedMoEExpertsModular):
    def __init__(self, moe_config, quant_config, max_num_tokens=None, num_dispatchers=None):
        super().__init__(moe_config, quant_config, max_num_tokens, num_dispatchers)
        self.is_fp8 = False
        self.is_int4 = False
        self.is_mxfp4 = False
        self.is_block_fp8 = False # 新增：标识 Block FP8 模式，默认关闭
        self.is_mxfp8 = False
```

```
        self.fused_moe_impl: XpuFusedMoe | None = None

    def apply(self, ...):
        # ... 守卫条件不变 ...
        if self.fused_moe_impl is None:
            topk = topk_ids.size(-1)
            self.fused_moe_impl = XpuFusedMoe(
                ...
                is_fp8=self.is_fp8,
                is_int4=self.is_int4,
                is_mxfp4=self.is_mxfp4,
                is_mxfp8=self.is_mxfp8,
                is_block_fp8=self.is_block_fp8, # 传递 Block FP8 标志给底层内核
            )
        self.fused_moe_impl.apply(...)

class XPUExpertsBlockFp8(XPUExperts):
    """Block FP8 MoE 专家实现，适用于 KFP8Static128BlockSym 权重量化。"""
    def __init__(self, moe_config, quant_config, max_num_tokens=None, num_dispatchers=None):
        super().__init__(moe_config, quant_config, max_num_tokens, num_dispatchers)
        self.is_block_fp8 = True # 开启 Block FP8 模式

    @staticmethod
    def _supports_quant_scheme(weight_key: QuantKey | None, activation_key: QuantKey | None)
    -> bool:
        # 声明支持的量化组合：权重为静态 128 块对称 FP8，激活为动态 128 对称 FP8
        SUPPORTED_W_A = [
            (kFp8Static128BlockSym, kFp8Dynamic128Sym),
        ]
        return (weight_key, activation_key) in SUPPORTED_W_A
```

评论区精华

gemini-code-assist[bot] 指出 apply 方法中直接访问 self.is_block_fp8 可能导致其他子类（如 XPUExpertsFp8）的 AttributeError，建议使用 getattr 提供默认值。不过提交者已在基类 __init__ 中初始化了该属性，因此该问题实际上已被解决，但讨论未进一步展开。最终获得了 xinyu-intel 和 jikunshang 的批准。

- is_block_fp8 属性安全性 (correctness): 提交者已在基类 __init__ 中初始化 self.is_block_fp8 = False，因此无需使用 getattr；问题已在代码中解决。

风险与影响

- 风险：回归风险：由于新增的 is_block_fp8 属性在基类中已初始化为 False，且 oracle 中仅追加了返回列表的末尾，不会影响现有 XPU MoE 专家类的加载顺序，回归风险较低。但未包含针对 Block FP8 的单元测试，仅依赖 CI 中的模型运行验证，可能遗漏边界情况。兼容性风险：XpuFusedMoe 内核需要支持 is_block_fp8 参数，若底层 vllm_xpu_kernels 版本不匹配则可能出错。

- 影响：用户影响：XPU 用户现在可以使用 --quantization fp8 运行为 Block FP8 量化的 MoE 模型（如 Qwen3-30B-A3B）。 系统影响：新增的类对原有 XPU MoE 流水线无侵入，仅增加了内核配置选项。 团队影响：Intel XPU 团队需确保 vllm_xpu_kernels 中 XpuFusedMoe 支持 is_block_fp8 参数。
- 风险标记：缺少单元测试覆盖，依赖底层内核版本

关联脉络

- 暂无明显关联 PR