PR #52368 完整报告

vllm-project/vllm

[Refactor] Simplify B12X linear kernels and warmup

合并时间：2026-08-18 00:45

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/52368

执行摘要

- 一句话：整合 B12X 线性内核与 warmup，统一为 provider 单元机制
- 推荐动作：值得精读，尤其是 provider 模式的设计：把“模型层挂载内核”和“warmup 编排”解耦，B12xWarmupUnit 以闭包捕获编译上下文、按 key 去重，是处理多格式 JIT 预热的可复用模式。对后续新增 B12X 算子或量化内核的贡献者是好模板；但建议合并前对照旧测试逐一确认行为等价性，并关注 nightly Spark job 是否真正跑满 4 个 SM12x 硬件用例。

功能与动机

PR body 声明目标是 'consolidate the B12X scaled-MM linear implementations and tests'、'replace per-format model scans with shared warmup units'、'simplify internal B12X lifecycle checks'。仓库无关联 Issue；动机来自维护成本：五个 warmup 函数各自扫一遍模型、每个内核用不同布尔标志记录生命周期，重复代码累积明显。PR 同时自述 'This is a refactor only; no kernel selection or supported-format change is intended.'

实现拆解

1. 合并 FP8 scaled-MM 内核文件：新增 vllm/model_executor/kernels/linear/scaled_mm/b12x.py（+339 行），把 b12x_block.py 的 B12xFp8BlockScaledMMKernel、_run_b12x_fp8_block_scaled_mm 与 b12x_tensor.py 的 B12xTensorFP8ScaledMMLinearKernel、_apply_b12x_tensor_fp8_packed_linear 收拢在同一文件；同时删除 b12x_block.py（-214 行）和 b12x_tensor.py（-238 行）。原因：两个 scaled-MM 内核共享同样的平台 / 形状校验骨架，拆在两个文件导致 is_supported、can_implement 反复铺垫重复代码。
2. 用 provider 挂载点统一生命周期：每种 B12X 线性内核的 process_weights_after_loading 末尾改为 layer.b12x_warmup_provider = self，block-FP8、tensor-FP8、MXFP8、NVFP4、MXFP4 五个内核统一用这一个属性替代原来的 b12x_block_fp8_linear、b12x_nvfp4_linear、b12x_mxfp4_linear 布尔标志与 b12x_*_packed_weight 存在性探测。每个内核新增 get_b12x_warmup_unit(layer, token_counts, output_dtype)，返回 B12xWarmupUnit（name/key/compile 三元组），key 由设备、形状、dtype 组成，compile 闭包内含各格式专属的 GEMM 预热调用（block-FP8 的 mm_block_fp8、MXFP8 的 mm + expected_m、NVFP4/MXFP4 的 mm_nvfp4/mm_mxfp4）。

3. 把 warmup 从“按格式分发”改成“按 provider 收集”：vllm/model_executor/warmup/b12x_warmup.py 删掉对 5 个 warmup_b12x_*_linear 函数的集中调用清单，改为 _collect_warmup_units 单次遍历 model.modules()，凡是层上挂有 provider 就生成 unit，再按 unit.key 去重；_compile_warmup_units 统一在 torch.inference_mode() 下调用 unit.compile() 并按 unit.name 统计。附带收益：旧实现对每个格式各自扫一遍 model，新实现只需一次遍历；日志里“实测多少次 GEMM”的口径从“签名×token 数”变为“签名数”。
4. 顺手做的小清理：mxfp8/b12x.py 删掉 _b12x_mxfp8_expected_m 辅助函数并内联为 max(1, int(tokens))，_apply_b12x_mxfp8_packed_linear 从 getattr + RuntimeError 防护收紧为直接访问 layer.b12x_mxfp8_packed_weight；vllm/utils/b12x.py（+9/-2）完善 B12xWarmupUnit 数据类 / 导出，构成 provider 与 warmup 编排之间的契约。
5. 测试与 CI 配套：原 test_b12x_mxfp8_linear.py 重命名为 test_b12x_linear.py，并把 NVFP4、MXFP4 两个测试文件（共 -445 行）合进去，改用参数化 test_b12x_backend_registration_priority_and_selection 覆盖 5 类内核的注册优先级、降级顺序、is_supported/can_implement 路径；test_b12x_warmup.py 重写为测试 provider 收集去重、token_counts 覆盖 serving 形状以及各格式 warmup unit 的编译调用（+166/-60）；.buildkite/test_areas/kernels.yaml +22 行新增 DGX Spark nightly job，跑 B12X linear、warmup 与 block-FP8 真机用例；tests/kernels/quantization/test_block_fp8.py 仅调整导入路径。

关键文件：

- vllm/model_executor/kernels/linear/scaled_mm/b12x.py（模块 线性内核；类别 source；类型 core-logic；符号 _run_b12x_fp8_block_scaled_mm, B12xFp8BlockScaledMMKernel, B12xTensorFP8ScaledMMLinearKernel, is_supported）：新建的 B12X FP8 scaled-MM 单一入口，合并了原先 b12x_block.py 与 b12x_tensor.py 的 block-FP8 和 tensor-FP8 两类内核，并统一挂载 b12x_warmup_provider、新增 get_b12x_warmup_unit。
- vllm/model_executor/warmup/b12x_warmup.py（模块 预热编排；类别 source；类型 core-logic；符号 _collect_warmup_units, _compile_warmup_units, b12x_warmup）：warmup 编排逻辑重写：从集中调用 5 个按格式 warmup 函数改为单次遍历模型收集 provider 单元并按 key 去重，是行为收敛的关键承载文件。
- vllm/model_executor/kernels/linear/mxfp8/b12x.py（模块 线性内核；类别 source；类型 core-logic；符号 B12xMxfp8LinearKernel, process_weights_after_loading, get_b12x_warmup_unit, compile）：MXFP8 内核在删除独立 warmup 函数后同样挂接 provider 机制，并顺带内联掉 _b12x_mxfp8_expected_m、收紧 packed weight 访问，是“减重复 + 内联”的典型代表。
- tests/model_executor/kernels/test_b12x_linear.py（模块 线性内核测试；类别 test；类型 test-coverage；符号 test_b12x_backend_registration_priority_and_selection, test_b12x_backend_maps_mxfp8_kernel, test_b12x_backend_maps_tensor_fp8_kernel, test_b12x_fp8_fallback_priority）：由 test_b12x_mxfp8_linear.py 重命名而来，吸收 NVFP4 / MXFP4 两个测试文件，用参数化测试统一覆盖 5 类 B12X 内核的注册优先级与选择逻辑，是测试收敛的核心。

- tests/model_executor/test_b12x_warmup.py（模块 预热测试；类别 test；类型 test-coverage；符号 test_b12x_warmup_token_counts_cover_serving_regimes, test_b12x_linear_warmup_skips_unused_provider, test_b12x_warmup_units_cover_token_counts, test_b12x_mxfp8_warmup_unit）：重写为验证新的 provider 收集、去重与 token_counts 覆盖 serving 形状的测试，直接支撑 warmup 重构的正确性。
- vllm/utils/b12x.py（模块 工具契约；类别 source；类型 data-contract；符号 B12xWarmupUnit）：定义 / 完善 B12xWarmupUnit（name/key/compile）这一核心契约类型，provider 与 warmup 编排器均依赖它。
- .buildkite/test_areas/kernels.yaml（模块 CI 配置；类别 config；类型 configuration）：新增 DGX Spark nightly job，专门覆盖 B12X linear、warmup 与 block-FP8 真机用例，补足本 PR 在普通 CI 上跳过的 SM12x 硬件覆盖。
- vllm/model_executor/kernels/linear/__init__.py（模块 线性内核；类别 source；类型 entrypoint）：调整 B12X 内核的导入路径，适配 b12x_block.py / b12x_tensor.py 被合并为单一 b12x.py 后的模块结构。
- tests/model_executor/kernels/test_b12x_nvfp4_linear.py（模块 线性内核测试；类别 test；类型 test-coverage；符号 test_b12x_backend_maps_nvfp4_kernel, test_b12x_nvfp4_fallback_priority, test_b12x_nvfp4_explicit_backend_selects_native_kernel, test_warmup_b12x_nvfp4_dedupes_weight_signatures）：删除并合并进统一的 test_b12x_linear.py，原先的 NVFP4 专用断言由参数化用例覆盖，减少重复维护。
- tests/model_executor/kernels/test_b12x_mxfp4_linear.py（模块 线性内核测试；类别 test；类型 test-coverage；符号 test_b12x_backend_maps_mxfp4_kernel, test_b12x_mxfp4_fallback_priority, test_b12x_mxfp4_requires_dynamic_activations, test_warmup_b12x_mxfp4_dedupes_weight_signatures）：删除并合并进统一的 test_b12x_linear.py，MXFP4 动态激活要求与 blockscaled gemm 调用断言由参数化用例覆盖。
- tests/kernels/quantization/test_block_fp8.py（模块 量化测试；类别 test；类型 test-coverage）：仅调整导入路径，适配 B12X block-FP8 内核从 b12x_block.py 迁入 b12x.py 的模块变化。

关键符号：get_b12x_warmup_unit, process_weights_after_loading, _collect_warmup_units, _compile_warmup_units, b12x_warmup, _run_b12x_fp8_block_scaled_mm, _apply_b12x_tensor_fp8_packed_linear, _apply_b12x_mxfp8_packed_linear

关键源码片段

vllm/model_executor/warmup/b12x_warmup.py

warmup 编排逻辑重写：从集中调用 5 个按格式 warmup 函数改为单次遍历模型收集 provider 单元并按 key 去重，是行为收敛的关键承载文件。

```
from collections import Counter
from collections.abc import Iterable
from typing import TYPE_CHECKING
```

```
import torch

from vllm.logger import init_logger
from vllm.platforms import current_platform
from vllm.utils.b12x import B12xWarmupUnit, b12x_warmup_token_counts

if TYPE_CHECKING:
    from vllm.v1.worker.gpu_worker import Worker

logger = init_logger(__name__)


# 新逻辑只遍历一次模型：每个 layer 若挂着 b12x_warmup_provider，
# 就让它自报 warmup 单元，并按 unit.key 去重，避免同一 GEMM 签名重复编译。
def _collect_warmup_units(
    model: torch.nn.Module,
    token_counts: tuple[int, ...],
    output_dtype: torch.dtype,
) -> Iterable[B12xWarmupUnit]:
    units: dict[object, B12xWarmupUnit] = {}
    for layer in model.modules():
        provider = getattr(layer, "b12x_warmup_provider", None)
        get_unit = getattr(provider, "get_b12x_warmup_unit", None)
        if not callable(get_unit):
            continue
        unit = get_unit(layer, token_counts, output_dtype)
        assert isinstance(unit, B12xWarmupUnit)
        units.setdefault(unit.key, unit)
    return units.values()


# 统一在 inference_mode 下执行各单元的 compile 闭包，并按单元 name 统计，
# 日志口径从旧实现的“签名 x token 数”变为“签名数”。
def _compile_warmup_units(units: Iterable[B12xWarmupUnit]) -> Counter[str]:
    warmed: Counter[str] = Counter()
    with torch.inference_mode():
        for unit in units:
            unit.compile()
            warmed[unit.name] += 1
    if warmed:
        torch.accelerator.synchronize()
    return warmed


def b12x_warmup(worker: "Worker", cudagraph_capture_sizes: list[int]) -> None:
    # B12X 后端只存在于 CUDA + SM12x 平台，非该平台直接跳过，不再依赖
    # 各格式 warmup 函数内部各自做一遍平台判断。
    if not current_platform.is_cuda():
        return
```

```
    if not current_platform.is_device_capability_family(120):
        return
    output_dtype = getattr(
        getattr(worker, "model_config", None), "dtype", torch.bfloat16,
    )
    if output_dtype not in (torch.bfloat16, torch.float16):
        output_dtype = torch.bfloat16
    token_counts = b12x_warmup_token_counts(
        max_tokens=worker.scheduler_config.max_num_batched_tokens,
        cudagraph_capture_sizes=cudagraph_capture_sizes,
    )
    units = _collect_warmup_units(worker.get_model(), token_counts, output_dtype)
    for name, count in _compile_warmup_units(units).items():
        logger.info_once(
            "Warmed up %d B12X %s linear GEMM signatures.", count, name,
        )
```

vllm/model_executor/kernels/linear/mxfp8/b12x.py

MXFP8 内核在删除独立 warmup 函数后同样挂接 provider 机制，并顺带内联掉 _b12x_mxfp8_expected_m、收紧 packed weight 访问，是“减重复 + 内联”的典型代表。

```
class B12xMxfp8LinearKernel(Mxfp8LinearKernel):
    """ModelOpt MXFP8 linear through the native b12x SM120 dense GEMM path."""

    def process_weights_after_loading(self, layer: torch.nn.Module) -> None:
        # 校验权重格式后，pack 成 B12X 原生布局，并把原 weight / weight_scale
        # 置空以释放显存；最后挂上 warmup provider，由编排器统一预热。
        weight = layer.weight.data
        assert weight.dtype == MXFP8_VALUE_DTYPE
        assert weight.ndim == 2
        assert in_features % MXFP8_BLOCK_SIZE == 0
        # ... pack_weight 与 replace_parameter 等原有逻辑保持不变 ...
        layer.b12x_warmup_provider = self

    def get_b12x_warmup_unit(
        self,
        layer: torch.nn.Module,
        token_counts: tuple[int, ...],
        output_dtype: torch.dtype,
    ) -> B12xWarmupUnit:
        packed_weight = layer.b12x_mxfp8_packed_weight
        device = torch.device(packed_weight.weight.values.device)

        # MXFP8 的 warmup 与 block-FP8 不同：直接调 mxfp8.mm 而非 mm_block_fp8，
        # expected_m 用 token 数兜底为 1，避免空输入时触发 shape 断言。
        def compile() -> None:
            mxfp8 = _import_b12x_mxfp8()
            assert mxfp8 is not None
            for tokens in token_counts:
```

```
            source = torch.zeros(
                (tokens, int(packed_weight.in_features)),
                dtype=output_dtype,
                device=device,
            )
            mxfp8.mm(
                source,
                packed_weight,
                expected_m=max(1, int(tokens)),
                stream=current_stream().cuda_stream,
            )

        return B12xWarmupUnit(
            name="MXFP8",
            key=(
                type(self),
                device,
                int(packed_weight.in_features),
                int(packed_weight.padded_in_features),
                int(packed_weight.out_features),
                output_dtype,
            ),
            compile=compile,
        )
```

评论区精华

refactor PR 引发的实质讨论很少：lukealonso 只留了一句全局认可，claude 机器人提示手动 review 流程，作者用 /ci run 触发 Buildkite CI #83957。没有 reviewer 对合并行为等价性提出质疑，也没有任何 review comment 要求改动，最终在 CI 通过后由 vllm-bot 合入。

- 整体重构验收 (design): 未产生任何代码修改要求，按原样合入。
- 草稿声明与人工复核要求 (question): 无实质讨论；按常规流程通过 CI 后由 vllm-bot 合入。
- CI 触发与构建状态 (other): CI 通过并合入。

风险与影响

- 风险：
  1. 行为等价性缺少真机回归：作者自述只跑了 35 个用例、4 个真实 block-FP8 硬件用例被跳过，依赖新增的 DGX Spark nightly job 兜底；合并 / 删除约 400 行源码与 3 个测试文件后，在没有 SM12x 真机的情况下完成合入，存在未暴露的 kernel 调用差异风险。
  2. 生命周期契约变更：layer.b12x_warmup_provider 取代了旧的 b12x_*_linear 布尔标志与 b12x_*_packed_weight 探测，任何外部代码依赖这些属性将失效；本 PR 同步清理了仓库内全部引用点，但属于对外不可见契约变更。
  3. 异常路径收紧：mxfp8/b12x.py 中 _apply_b12x_mxfp8_packed_linear 从 getattr + RuntimeError（带提示文案）改为直接属性访问，若 process_weights_after_loading 未执行会抛 AttributeError，排查体验变差。

4. warmup 统计口径变化：日志中“Warmed up N signatures”的 N 从“签名×token 数”变为“签名数”，依赖日志做监控或告警的团队需同步调整预期。
5. 测试文件大量合并：test_b12x_linear.py 一口气吸收 NVFP4/MXFP4 两个文件的用例并转为参数化，覆盖顺序与断言方式变化较大，存在误删或弱化个别边界断言的风险。 - 影响：影响范围集中在 DGX Spark / GB10（SM120 Blackwell）用户的 B12X 后端路径，不影响其它平台：启动 warmup 从“五遍模型扫描”降为“一遍收集 + 去重编译”，可减少重复 JIT 触发与启动耗时；代码层面，b12x_warmup_provider 成为 B12X 内核的隐含接口，未来新增量化格式内核只需实现 get_b12x_warmup_unit，维护成本明显下降；团队侧，测试从 3 个文件收敛为 1 个参数化文件，并新增 nightly 真机覆盖，CI 结构更清晰。影响程度整体为中：纯重构、无用户可见行为变化，但涉及核心 kernel 生命周期契约的收敛。 - 风险标记：B12X 后端路径变更，生命周期契约变更，warmup 统计口径变化，缺少真机 SM12x 回归，轻量 review 后合并

关联脉络

- PR #52502 [Hardware][NVIDIA] Add GB10 fused-MoE fp8 tuning configs (E=256, E=512): GB10 即 DGX Spark 的 Blackwell 芯片，与本 PR 的 B12X（SM12x）FP8 GEMM 属于同一硬件与同一 FP8 内核优化方向，可视为 B12X 后端持续投入的并行部分。