PR #48774 完整报告

vllm-project/vllm

[Perf] Tune LL BF16 Router GEMM

合并时间：2026-07-27 22:25

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/48774

执行摘要

- 一句话：调优 LL BF16 Router GEMM 分发启发式用于 DeepSeek/GLM 关键模型
- 推荐动作：值得精读，尤其是对 GPU GEMM 调优感兴趣的工程师。PR 展示了如何结合端到端测量和架构感知配置来优化内核分发，以及如何用动态形状推导替代静态枚举以增强模型兼容性。未来可基于此构建自动调优器。

功能与动机

Router GEMM 若不仔细处理会消耗大量性能；之前使用的基于 PTX 的 ad-hoc 内核局限于单一或极少数形状，难以扩展至新模型。PR #42562 引入了通用的 cuteDSL 实现，但默认配置未充分调优。由于 PDL 重叠导致微基准无法准确反映端到端行为，因此需要基于端到端测量来调优分发决策。

实现拆解

1. 动态形状推导：在 kernel_warmup.py 中新增 _ll_bf16_router_shapes_from_model 函数，通过遍历模型模块中的所有 GateLinear 层，动态收集 bf16 权重的 (K, N) 形状（要求 K 为 8 的倍数）。_warmup_ll_bf16_router_gemm 改为接收模型对象并调用该函数，替代了原先的静态 _LL_BF16_WARMUP_MODEL_SHAPES 元组。
2. 移除 MoE 门控：在 kernel_warmup 的调用点上，移除 worker.model_config.is_moe 条件，改为始终检查是否存在 GateLinear 形状。非 MoE 模型或无形状的模型将自动跳过 warmup 并记录跳过日志。
3. 架构特定调优配置：在 ll_bf16.py 中删除原有的 _TUNED_DOTPROD_MAX_M（对 (7168,256) 特例）和 _TUNED_CONFIGS，改为引入两组独立的调优字典：_SM100F_TUNED_DOTPROD_BS / _SM100F_TUNED_SPLITK_CONFIGS 和 _SM90_TUNED_DOTPROD_BS / _SM90_TUNED_SPLITK_CONFIGS，分别覆盖 Blackwell 和 Hopper。
4. 分发逻辑重构：新增 _arch_tuned_configs() 函数，根据当前 GPU 架构返回对应的调优配置字典。LLBf16Gemm.dispatch 方法改为从该函数获取配置：若 M <= 4（默认点积后端最大 M）或 K < 2048，则使用点积后端并应用调优的 block size bs；否则使用 split-K 后端并应用调优的 (split_k, num_stages)。
5. 配套调整：统一了对 dotprod_max_m 的判断逻辑，使其与调优配置解耦，并为新增形状添加了注释说明。

关键文件：

- vllm/model_executor/warmup/kernel_warmup.py（模块 内核预热；类别 source；类型 data-contract；符号 _warmup_ll_bf16_router_gemm, _ll_bf16_router_shapes_from_model）：实现了从模型动态推导路由形状，移除了静态形状列表和 MoE 门控，变更 warmup 条件。
- vllm/model_executor/kernels/linear/cute_dsl/ll_bf16.py（模块 BF16 路由 GEMM；类别 source；类型 data-contract；符号 _arch_tuned_configs）：重构调优配置表为 SM90/SM100F 特定表；新增 _arch_tuned_configs 函数；简化 dispatch 逻辑，移除特例。

关键符号：_ll_bf16_router_shapes_from_model, _warmup_ll_bf16_router_gemm, _arch_tuned_configs, LLBf16Gemm.dispatch

关键源码片段

vllm/model_executor/kernels/linear/cute_dsl/ll_bf16.py

重构调优配置表为 SM90/SM100F 特定表；新增 _arch_tuned_configs 函数；简化 dispatch 逻辑，移除特例。

```
# 默认配置常量
_DEFAULT_DOTPROD_BS = 128
_DEFAULT_DOTPROD_MAX_M = 4
_DEFAULT_SPLITK_CONFIG = (6, 4)

# SM100f (Blackwell) 特调配置
_SM100F_TUNED_DOTPROD_BS: dict[tuple[int, int], dict[int, int]] = {
    (6144, 256): {M: 256 for M in (1, 3, 4)},
}
_SM100F_TUNED_SPLITK_CONFIGS: dict[tuple[int, int], dict[int, tuple[int, int]]] = {
    (4096, 256): {**{M: (8, 5) for M in (5, 8)}, 9: (8, 2)},
    (7168, 256): {14: (8, 2)},
    (6144, 256): {M: (8, 2) for M in (9, 12, 16)},
    (7168, 384): {M: (7, 5) for M in (13, 16)},
}

# SM90 (Hopper) 特调配置
_SM90_TUNED_DOTPROD_BS: dict[tuple[int, int], dict[int, int]] = {
    (4096, 256): {M: 256 for M in (1, 3)},
    (7168, 384): {M: 256 for M in (1, 2)},
}
_SM90_TUNED_SPLITK_CONFIGS: dict[tuple[int, int], dict[int, tuple[int, int]]] = {
    (4096, 256): {**{M: (8, 2) for M in range(5, 8)}, **{M: (8, 5) for M in range(10, 12)}, **{M:
    (8, 2) for M in (13, 16)}, **{M: (8, 5) for M in (14, 15)}},
    (7168, 256): {8: (6, 5)},
    (6144, 256): {M: (8, 2) for M in (9, 11)},
    (7168, 384): {**{M: (8, 2) for M in (6, 8, 12)}, **{M: (7, 5) for M in (7, 9, 10, 11, 13, 14, 15,
    16)}},
}
```

```
def _arch_tuned_configs() -> tuple[
    dict[tuple[int, int], dict[int, int]],
    dict[tuple[int, int], dict[int, tuple[int, int]]],
]:
    """根据 GPU 架构返回调优后的 dotprod block size 和 split-K 配置字典。"""
    from vllm.platforms import current_platform

    if current_platform.is_device_capability_family(100):
        return _SM100F_TUNED_DOTPROD_BS, _SM100F_TUNED_SPLITK_CONFIGS
    if current_platform.is_device_capability(90):
        return _SM90_TUNED_DOTPROD_BS, _SM90_TUNED_SPLITK_CONFIGS
    return {}, {}


class LLBf16Gemm:
    # ... 省略 __init__ 和缓存 ...

    def dispatch(self, *, M: int, K: int, N: int) -> "CompileKey":
        """根据 M, K, N 和 GPU 架构选择 dotprod 或 split-K 后端。"""
        tuned_bs, tuned_splitk = _arch_tuned_configs()
        if M <= _DEFAULT_DOTPROD_MAX_M or K < 2048:
            # 使用 dotprod 后端，应用调优的 block size（若无则使用默认值 128）
            bs = tuned_bs.get((K, N), {}).get(M, _DEFAULT_DOTPROD_BS)
            return self.CompileKey(backend="dotprod", M=M, K=K, bs=bs)

        # 使用 split-K 后端，应用调优的 (split_k, num_stages)
        split_k, num_stages = tuned_splitk.get((K, N), {}).get(
            M, _DEFAULT_SPLITK_CONFIG
        )
        return self.CompileKey(
            backend="splitk", split_k=split_k, num_stages=num_stages
        )
```

评论区精华

本 PR 未引发实质性技术讨论。mergify 机器人报告了合并冲突，作者 rebase 后 CI 通过，最终由 mgoin 批准合并。

- 暂无高价值评论线程

风险与影响

- 风险：
  - 动态形状依赖模型结构：形状推导依赖于 GateLinear 模块的存在及权重属性；若模型有非标准路由实现，可能无法识别形状导致 warmup 跳过或编译遗漏。
  - 架构配置维护负担：新增的 SM90/SM100F 调优表需要针对每个新架构或新模型手动进行端到端测量，缺少自动调优器。未来 GPU 架构可能回退到默认配置，性能可能未达最优。

- 测试覆盖不足：无新增测试，难以验证边缘情况（如非 8 倍数的 K 尺寸、非 MoE 模型行为等）。
- 条件变更风险：移除了 is_moe 门控，但通过形状检查确保安全；若模型意外包含 GateLinear 但非 MoE（如推理时禁用 MoE），可能触发预期外的 warmup。
- 影响：影响范围限于使用 ll_bf16_gemm 内核的 MoE 模型（DeepSeek、GLM、Inkling 等）。对于这些模型，端到端吞吐量预计提升 1-6%（随并发度变化）。不涉及 API 变更或用户可见行为改变。warmup 时长因动态形状推导可能略有增加，但总体影响较小。团队需维护架构特定配置表，但代码结构清晰，易于扩展。
- 风险标记：缺少测试覆盖，配置维护负担，动态形状依赖模型结构

关联脉络

- PR #42562 [Perf] Add cuteDSL-based LL bf16 GEMM kernel: 引入了 cuteDSL 实现，本 PR 在此基础上进行调优。
- PR #49659 [Perf] Skip ll_bf16 router GEMM warmup for non-MoE models: 基于本 PR 的动态形状推导，进一步优化非 MoE 模型的 warmup 跳过逻辑。