

PR #54261 完整报告

vllm-project/vllm

[Kimi-K3][Perf] Make native CUDA AttnRes the SM100 default

合并时间: 2026-09-01 07:18

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/54261>

执行摘要

- 一句话: SM100 上 Kimi-K3 AttnRes 默认切原生 CUDA, 最高提速 1.72x
- 推荐动作: 值得精读。该 PR 是「让原生内核从小覆盖变体扩展为架构默认」的完整范例: 编译期特化 (输出归一化关闭路径在编译期剔除 smem/barrier/ 算术)、PDL 等待的延迟最小化、运行时与编译期双轨折叠 (STATIC_COMMON_PATH)、以及调度门控对 stride 与边界的最严格校验, 都是 SM100 内核工程中可复用的设计决策。建议重点关注 `attn_res_kernel.cu` 中 PDL 纪律与 block 写回逻辑、`attn_res.py` 中默认路径判定条件, 以及 `test_sm100_variants_do_not_fall_back_to_triton` 的 monkeypatch 测试手法。若团队后续要做其他架构 (ROCm/XPU) 的同类默认化, 可参照其「API 契约先行 + 门控收紧 + 基准可复现 + 回退保底」的推进节奏。

功能与动机

Kimi-K3 的 AttnRes 是 dense 输入路径上的热点算子: 每个 token 需在「最多 8 个 block 残差 + 当前层残差」上执行 RMSNorm、与 qk 权重做点积 softmax 后加权求和。此前原生 CUDA 内核只覆盖有限变体 (要求 delta 与输出归一化都存在、block 数大于 0), 其余变体 (block 边界写回、无输出归一化、单 token 小 batch) 均落到 Triton。PR body 明确目标为 “Make the native CUDA Kimi-K3 AttnRes implementation the default for dense hidden_size=7168 inputs on SM100”, 并给出量化动机: 单 token 多 block 是投机解码 / 多 token 验证的热路径, Triton 随 block 行数线性增长 (1/4/8 block 为 10.24/12.29/14.34 us), 而原生内核几乎平坦 (8.19/8.22/10.24 us)。PR body 还专门说明与 #53524 (AttnRes PDL handoff 并入本 PR) 和 #50760 (仅改 warmup profiles) 的职责划分, 以避免重复工作。

实现拆解

整个变更按「API 契约 → 内核扩展 → 调度门控 → 测试与基准」四步展开:

1. 扩展算子 API 契约: `csrc/libtorch_stable/ops.h` 与 `torch_bindings.cpp` 中将 `kimi_k3_attn_res` 的 `delta`、`output_norm_weight` 从必填改为 `std::optional<Tensor>` (TorchScript schema 中为 `Tensor?`), `blocks` 从只读 `const&` 改为可写 `Tensor!`, 并新增 `block_write_idx` 参数 (-1 表示不写回 block bank); `vllm/_custom_ops.py` 的 Python 封装同步改为 `delta: torch.Tensor | None`、`output_norm_weight: torch.Tensor | None` 并透传 `block_write_idx`。这一步为内核覆盖「可选前缀相加、可选输出归一化、block-bank 写回」三类变体提供了入口。

2. 内核功能扩展 (`csrc/libtorch_stable/kimi_k3/attn_res_kernel.cu`, +104/-77) : 模板参数从 `HAS_DELTA/OUTPUT_NORM_IN_SMEM` 重组为 `STATIC_COMMON_PATH/HAS_OUTPUT_NORM`; `has_delta` 与 `write_idx` 改为运行时变量, 同时保留编译期折叠的公共路径 (`STATIC_COMMON_PATH`, 此时强制要求 `HAS_OUTPUT_NORM`)。输出归一化改为编译期特化, 关闭路径在编译期剔除 `output_norm_buf` 的 shared memory、`mbarrier` 与乘法算术。PDL 等待从内核开头的无条件 `cudaGridDependencySynchronize()` 推迟到「第一条 producer 依赖的 prefix/delta TMA」之前, 并在全部 producer TMA 提交后调用 `cudaTriggerProgrammaticLaunchCompletion()` 显式唤起消费者。消费侧新增按 `write_idx` 将结果行写回 `block_res` block bank 的逻辑。
3. 放宽默认调度门控 (`vllm/models/kimi_k3/nvidia/ops/attn_res.py`, +9/-8) : 原条件要求 `delta is not None`、`output_norm_weight is not None`、`num_blocks > 0` 且 `block_write_idx < 0`; 新条件放宽为 dense 行布局 (`prefix.stride(0) == hidden_size` 及 `delta` 同理)、`0 <= num_blocks <= 8`、`num_blocks <= blocks.shape[1]`、`-1 <= block_write_idx < blocks.shape[1]`, 并要求 SM100 家族 + `torch.ops._C` 中存在该 op。不满足时继续回退 Triton。行布局与边界双重校验是防止内核越界或错误原地写回的关键。
4. 测试与可复现基准配套: `tests/models/kimi_k3/test_attn_res.py` 新增 `test_sm100_variants_do_not_fall_back_to_triton`, 用 `FailingKernel` (`__getitem__` 即抛 `AssertionError`) `monkeypatch_attn_res_kernel`, 参数化覆盖 0/4/8 blocks、有无 delta、有无输出归一化、是否写回, 确保所有变体都不再回退 Triton; 参数表新增 `nvidia-single-token` (1 token、4 blocks) 小 batch 用例。新增 `benchmarks/kernels/benchmark_kimi_k3_attn_res.py` (235 行), 先用 `torch.testing.assert_close` 对 0/1/4/8 blocks 做 CUDA-vs-Triton 数值对齐校验, 再以 `triton.testing.do_bench` 输出 p20/p50/p80 并落盘含设备元数据的 JSON。

关键文件:

- `csrc/libtorch_stable/kimi_k3/attn_res_kernel.cu` (模块 内核实现; 类别 source; 类型 core-logic; 符号 `attn_res_fwd_online_v2_kernel`) : 核心内核文件: 模板参数重组 (`STATIC_COMMON_PATH/HAS_OUTPUT_NORM`)、PDL 等待延迟到首个 producer 依赖 TMA、block-bank 写回、输出归一化编译期特化全部发生在这里, 是性能收益与绝大多数风险来源。
- `vllm/models/kimi_k3/nvidia/ops/attn_res.py` (模块 算子调度; 类别 source; 类型 core-logic; 符号 `attn_res`) : 默认路径调度门控所在: 决定哪些形态走原生 CUDA、哪些回退 Triton, 是本次「默认化」的直接承载点; stride 与边界校验的严谨程度直接决定内核安全性。
- `benchmarks/kernels/benchmark_kimi_k3_attn_res.py` (模块 性能基准; 类别 source; 类型 benchmark; 符号 `Inputs`, `Result`, `parse_args`, `make_inputs`) : 新增的可复现 CUDA-vs-Triton 基准 (235 行) : 先做数值对齐校验再计时, 并落盘含设备元数据的 JSON, 为后续内核演进提供统一度量工具。
- `tests/models/kimi_k3/test_attn_res.py` (模块 回归测试; 类别 test; 类型 test-coverage ; 符号 `test_sm100_variants_do_not_fall_back_to_triton`, `FailingKernel`, `getitem`) : 新增 `test_sm100_variants_do_not_fall_back_to_triton`, 用 `FailingKernel` `monkeypatch` 验

证 SM100 下所有变体都不回退 Triton，是防止调度回归的关键测试。

- `csrc/libtorch_stable/ops.h`（模块 算子接口；类别 source；类型 data-contract；符号 `kimi_k3_attn_res`）：算子 C++ 声明变更：delta/output_norm_weight 改为 optional，blocks 改为可写引用，新增 `block_write_idx`，是内核新能力的 API 契约源头。
- `csrc/libtorch_stable/torch_bindings.cpp`（模块 算子绑定；类别 source；类型 data-contract；符号 `kimi_k3_attn_res`）：TorchScript op schema 与 C++ 声明对齐：Tensor? delta、Tensor! blocks、Tensor? output_norm_weight 与 int block_write_idx，保证 Python 侧可安全传 None 与原地写回。
- `vllm/_custom_ops.py`（模块 算子封装；类别 source；类型 data-contract；符号 `kimi_k3_attn_res`）：Python 封装同步扩展：delta 与 output_norm_weight 类型改为 Optional，新增 `block_write_idx` 透传，是模型代码与原生 op 之间的桥梁。

关键符号：attn_res, kimi_k3_attn_res, attn_res_fwd_online_v2_kernel, test_sm100_variants_do_not_fall_back_to_triton, validate, launch_triton, launch_cuda, main

关键源码片段

`csrc/libtorch_stable/kimi_k3/attn_res_kernel.cu`

核心内核文件：模板参数重组（STATIC_COMMON_PATH/HAS_OUTPUT_NORM）、PDL 等待延迟到首个 producer 依赖 TMA、block-bank 写回、输出归一化编译期特化全部发生在这里，是性能收益与绝大多数风险来源。

```
// csrc/libtorch_stable/kimi_k3/attn_res_kernel.cu（核心控制流重组）
// PDL 纪律：等待前序网格的操作被推迟到「第一条真正依赖 producer 的 TMA」之前，
// 而不是在内核开头无条件同步。对纯 block 输入（无 delta）只需在读到末行
// block 残差前同步一次；全部 producer TMA 提交后，再用
// cudaTriggerProgrammaticLaunchCompletion() 显式唤起消费者网格。
bool pdl_waited = false;
for (int tb = blockIdx.x; tb < TB; tb += num_ctas) {
    for (int ci = 0; ci < num_chunks; ci++, gci++) {
        // ... 先等待消费方释放 chunk 槽位 (mbarrier) ...
        const bf16_t* src = residual_addr(block_res, layer_res, ns + n, N, t,
                                           block_stride_m, block_stride_r, H);
        if (!has_delta && ns + n == N - 1 && !pdl_waited) {
            cudaGridDependencySynchronize(); // 前缀行依赖前序网格输出，首次读到才等
            pdl_waited = true;
        }
        cp_async_bulk(buf_ptr(slot), src, H * sizeof(bf16_t), plan.bar_ready[chunk_slot]);

        // delta 前缀相加变体：等待推迟到第一条 delta TMA 之前
        if (has_delta) {
            int prefix_n = N - 1 - ns;
            if (prefix_n >= 0 && prefix_n < an) {
                if (!pdl_waited) {
                    cudaGridDependencySynchronize();
                    pdl_waited = true;
                }
            }
        }
    }
}
```

```

    }
    cp_async_bulk(delta_buf_ptr(chunk_slot), delta + (long long)tb * H,
                  H * sizeof(bf16_t), plan.bar_ready[chunk_slot]);
    }
    }
}
}
cudaTriggerProgrammaticLaunchCompletion();

// 消费侧：前缀行既可能叠加 delta，也可能写回 block bank (write_idx >= 0)
int prefix_n = N - 1 - ns;
if (n == prefix_n && (has_delta || write_idx >= 0)) {
    if (has_delta) {
        // 把 delta 累加到 vp，然后再写回 layer_res
    }
    *reinterpret_cast<int4*>(layer_res + (long long)tb * H + h_base) = vp;
    if (write_idx >= 0) {
        *reinterpret_cast<int4*>(block_res + (long long)(tb / B) * block_stride_m
                                + write_idx * block_stride_r + h_base) = vp;
    }
}

// 输出归一化是编译期特化 (HAS_OUTPUT_NORM)：关闭路径在编译期直接剔除
// output_norm_buf 的 shared memory、mbarrier 与乘法算术，运行期零开销
if constexpr (HAS_OUTPUT_NORM) {
    // 不变权重复制只等待一次，之后所有 tb 共享同一份副本
    if (tb == blockIdx.x) {
        mbarrier_wait(plan.bar_output_norm_ready, 0);
    }
    float output_sq = output_sq_pair.x + output_sq_pair.y;
    // ... 计算 rsigma 后逐元素乘权重 ...
    float weight = __bfloat162float(output_norm_buf[h_base + j]);
    values[j] = __float2bfloat16(acc32[si * VEC + j] * output_rsigma * weight);
}

```

vllm/models/kimi_k3/nvidia/ops/attn_res.py

默认路径调度门控所在：决定哪些形态走原生 CUDA、哪些回退 Triton，是本次「默认化」的直接承载点；stride 与边界校验的严谨程度直接决定内核安全性。

```

# vllm/models/kimi_k3/nvidia/ops/attn_res.py (默认路径判定)
# 原生 CUDA 内核已覆盖 SM100 dense 输入的全部 AttnRes 变体：
# 0-8 个 block、可选 delta 前缀相加、可选输出归一化、block-bank 写回。
# 该 op 只在 CUDA >= 13 下编译，因此仅凭设备型号不足以判断其存在，
# 必须同时检查 torch.ops._C 中是否有对应符号。
assert norm_weight.stride(-1) == 1
assert qk_weight.stride(-1) == 1
assert output_norm_weight is None or output_norm_weight.stride(-1) == 1

if (hidden_size == 7168

```

```

and prefix.stride(0) == hidden_size # 行必须连续, 否则走 Triton
and (delta is None or delta.stride(0) == hidden_size)
and 0 <= num_blocks <= 8 # 内核只支持 0-8 个 block 行
and num_blocks <= blocks.shape[1]
and -1 <= block_write_idx < blocks.shape[1] # 写回索引越界直接拒绝
and current_platform.is_device_capability_family(100)
and hasattr(torch.ops._C, "kimi_k3_attn_res"):
return _custom_ops.kimi_k3_attn_res(
    prefix, delta, blocks, norm_weight, qk_weight, output_norm_weight,
    num_blocks, block_write_idx, eps, output_norm_eps)

```

非 SM100、行不连续或无原生 op 的实现仍走下方 Triton 回退路径

benchmarks/kernels/benchmark_kimi_k3_attn_res.py

新增的可复现 CUDA-vs-Triton 基准（235 行）：先做数值对齐校验再计时，并落盘含设备元数据的 JSON，为后续内核演进提供统一度量工具。

```

# benchmarks/kernels/benchmark_kimi_k3_attn_res.py
# 基准约定: delta 恒为零, 保证重复原地 launch 结果稳定;
# CUDA 侧使用 block_write_idx=-1, 避免 block bank 原地写回影响重复计时。
def launch_cuda(inputs: Inputs, output: torch.Tensor, num_blocks: int) -> None:
    torch.ops._C.kimi_k3_attn_res(
        inputs.prefix, inputs.delta, inputs.blocks, inputs.norm_weight,
        inputs.qk_weight, inputs.output_norm_weight, output,
        num_blocks, -1, EPS, EPS,
    )

def validate(inputs: Inputs, num_blocks: int) -> None:
    # 正确性校验先行: 同一输入分别走 Triton 与 CUDA, 对齐后才进入计时
    outputs = [torch.empty_like(inputs.prefix) for _ in range(2)]
    launch_triton(inputs, outputs[0], num_blocks)
    launch_cuda(inputs, outputs[1], num_blocks)
    torch.accelerator.synchronize()
    torch.testing.assert_close(outputs[1], outputs[0], atol=8e-2, rtol=3e-2)

def main() -> None:
    args = parse_args()
    if torch.cuda.get_device_capability()[0] != 10:
        raise SystemExit("The vLLM CUDA AttnRes kernel requires the SM100 family")
    if not hasattr(torch.ops._C, "kimi_k3_attn_res"):
        raise SystemExit("This vLLM build does not contain kimi_k3_attn_res")

    torch.manual_seed(0)
    check_inputs = make_inputs(5)
    for num_blocks in args.num_blocks:
        validate(check_inputs, num_blocks)

```

```

# 全部 (blocks, tokens) 组合下对两个后端分别 do_bench,
# 输出 p20/p50/p80 并把设备元数据与结果一起落盘为 JSON, 保证可复现
results = []
for num_blocks in args.num_blocks:
    for num_tokens in args.num_tokens:
        inputs = make_inputs(num_tokens)
        output = torch.empty_like(inputs.prefix)
        candidates = {
            "vllm_triton": partial(launch_triton, inputs, output, num_blocks),
            "vllm_cuda": partial(launch_cuda, inputs, output, num_blocks),
        }
        for backend, candidate in candidates.items():
            median_us, p20_us, p80_us = benchmark(
                candidate, args.warmup_ms, args.repeat_ms)
            results.append(Result(num_tokens, num_blocks, backend,
                                median_us, p20_us, p80_us))

```

评论区精华

本 PR 的 review 评论数为 0, 实质讨论主要来自 PR body 与 CI 触发记录:

- claude[bot] 提示该 PR 来自 fork, 自动 review 被禁用, 维护者可手动触发 @claude review; 最终未触发额外自动审核。
- 维护者 jeejeelee 直接 APPROVED 且未附评论, 说明对「原生内核成为 SM100 默认路径」的变更无新增异议。
- PR body 中最重要的设计协商是与 #53524/#50760 的职责边界: 作者声明 #53524 的 AttnRes PDL handoff 已并入本 PR 集成化改动, 该 PR 剩余独立范围是 C=1 MoE/router 工作; #50760 仅改 warmup profiles。二者都未把原生内核扩展到全部 Kimi-K3 AttnRes 调用变体, 因此无重复工作。
- PR body 同时披露 AI 辅助实现 (OpenAI Codex、ming@inference.ai 为 co-author), 并说明提交者逐行复核后才会合入。
- fork PR 自动 review 被禁用 (other): 保持人工审核流程, 由维护者 jeejeelee 直接批准。
- 维护者批准 (other): 批准合入 main。
- 与 #53524/#50760 的职责边界 (design): 职责边界清晰, 无重复工作, PDL handoff 由本 PR 统一承载。

风险与影响

- 风险:
 1. SM100 专属默认路径的回归面: 原生内核成为 SM100 家族 + CUDA ≥ 13 构建下 dense 输入的默认实现, 若内核在未覆盖形态 (如未来新的调用变体) 上行为异常, 将直接进入生产路径; 回退正确性依赖 attn_res.py 中 stride/ 边界检查的完备性。
 2. blocks 原地写回语义: ops.h 中 blocks 从 const& 改为可写 Tensor&, torch_bindings.cpp 标记为 Tensor!; 当 block_write_idx ≥ 0 时内核会直接改写 block bank 缓冲, 调用方若复用同一 blocks 张量可能观察到原地修改。测试已覆盖写回

变体，但生产路径中该语义需要调用方知晓。

3. 数值容差较宽松：CUDA-vs-Triton 与模型测试均使用 $\text{atol}=8\text{e-}2$, $\text{rtol}=3\text{e-}2$ 的 BF16 容差；模型评估（GSM8K/OCRBench）未显示回退，但 $n=1319$ 的采样方差约 ± 0.5 个点，对小数值回归的敏感度有限。
4. PDL 依赖 CUDA 13 语义：延迟 `cudaGridDependencySynchronize()` 与 `cudaTriggerProgrammaticLaunchCompletion()` 依赖 SM100 的 `programmatic dependent launch` 特性，PR 提供了 CUDA graph 捕获下的 profiler 数据，但该机制在 graph 回放与多 stream 场景下的边界行为仍依赖 CUDA 13 的稳定性。
5. 编译与维护成本：模板实参组合增加（`STATIC_COMMON_PATH` × `HAS_OUTPUT_NORM` 等），可能增大编译时间与二进制体积；基准脚本被 pre-commit 纳入检查，后续内核改动需同步维护。
6. 提交历史非线性：3 个提交中 2 个为 merge main 提交（91ddd3647f4d、2d2c75d2dfc7），非 rebase 干净历史，回溯粒度稍差。- 影响：影响范围集中在「NVIDIA SM100 + CUDA ≥ 13 + Kimi-K3 模型 + dense 行布局」这一组合：默认算子从 Triton 切换到原生 CUDA，小 batch ($M \leq 16$) 解码路径提速 1.20x-1.49x，单 token 4-block 场景达 1.49x；大 batch prefill (4096/8192 token) 提速 1.67x-1.72x。无输出归一化特化在大形状下额外带来最高 4.6% 收益（如 8-block、 $M=1024$ 从 40.86 us 降至 38.98 us）。对用户而言，Kimi-K3 在 B300/GB300 上的 TTFT/ITL 均受益，且五种拓扑（TP8、TEP8、DEP16、限并发、串行）评估均无精度回退；对团队而言，新增的基准脚本与「不回退 Triton」测试为后续内核演进提供了回归护栏，CI 需要能提供 SM100 设备的构建与测试资源。非 SM100 平台（ROCm、XPU、Intel GPU、老架构 NVIDIA）完全不受影响，仍走 Triton。- 风险标记：SM100 专属默认路径，内核原地写回 blocks，数值容差较宽松，PDL 依赖 CUDA 13 语义，AI 辅助实现需人工复核

关联脉络

- PR #54636 [Kimi Bug] Fix gdn build_attn_metadata 'KimiK3KDAMetadataBuilder' object has no attribute 'layer_names': 同为 Kimi-K3 后端的近期修复（gdn_attn.py），与本次 AttnRes 内核默认化同属 Kimi-K3 在 vLLM 上的收敛闭环。
- PR #54634 [K3 Bug] Fix Kimi-K3 RecoverSSM startup failure 'MambaAttentionBackendEnum.GDN_ATTN declares 4 states, but provides 2 state copy funcs': Kimi-K3 投机解码 /SSM 路径的近期缺陷修复，说明该模型家族的支撑面正在快速完善。
- PR #51705 [ROCm][MLA][DCP] Support causal multi-token verification: Kimi-K3 在 ROCm 侧的注意力 / 投机验证内核工作，与本次 NVIDIA SM100 内核默认化形成同一模型家族在异构平台上的对应演进。
- PR #53598 [ROCm][DSpark][DCP] Serve prefix cache hits under DCP for Kimi-K3: Kimi-K3 前缀缓存与 KV 路径修复，与 AttnRes 同属该模型 dense 推理链路上的热点。