

PR #53053 完整报告

vllm-project/vllm

[Kimi-K3] Extend GEMM-RS to GEMM-AR

合并时间: 2026-08-21 11:44

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/53053>

执行摘要

- 一句话: Kimi-K3 新增 GEMM-AR 融合内核, kernel 内完成 all-reduce
- 推荐动作: 值得精读。核心看点: (1) 用 CuTe DSL 的 `llvm.inline_asm` 封装 `multimem.st` 多播存储, 把 `collective` 语义融合进 GEMM 内核的实现手法; (2) 输出 `workspace` 复用带来的生命周期设计权衡与对应测试锚点; (3) 初始化失败优雅降级 (NVLink 缺失时回退普通 GEMM) 的容错模式。若后续计划支持非 NVLink 多播环境或消除输出 clone, 应在此 PR 基础上迭代。

功能与动机

PR body 明确说明这是对 #52079 的 GEMM-RS 的扩展, 并取代早前的 #52687: "Extends GEMM-RS, introduced by #52079, to GEMM-AR by replacing unicast store with `multimem.st`". 相对 #52687 的优势是 "Any value of M is supported -> no padding/input copy is required" 与 "Flag signalling/synchronization is fully done inside the kernel -> no flag reset + entry barrier is needed". 动机是让 non-SP 部署也能吃到融合聚合 GEMM 的性能红利: 微基准在 $M \geq 128$ 时约 1.0-1.21x, $M=32768$ 时达 1.21x; E2E 在并发 8 时 TTFT 降低 5.74%, 吞吐提升 1.22%。

实现拆解

整体分四步推进, 配套修改 9 个文件:

1. 内核层: RS 泛化为 RS/AR 双模式。 `vllm/models/kimi_k3/nvidia/ops/cute_dsl/gemm_rs.py` 重命名为 `gemm_rs_ar.py`, 新增 `@dsl_user_op` 的 `multimem_st_16B`, 通过 `llvm.inline_asm` 发射 `multimem.st.relaxed.sys.global.v4.f32` 多播存储指令; `Sm100GemmRsBF16` 泛化为 `Sm100GemmRsArBF16`, 构造参数新增 `all_reduce`。 kernel 内 `local_M` 在 AR 模式下取 `partial_uc.shape[0] // num_ranks` (全局行数), RS 模式仍取本 rank 的 `output.shape[0]`; epilogue 只在 RS 模式做 output 向量切分, AR 模式由通信 warp 把部分和通过 `multimem_st_16B` 多播写回对称内存 `workspace`, 从而把 all-reduce 语义融进 GEMM 内核。
2. 特性开关与初始化降级。 `vllm/envs.py` 新增 `VLLM_KIMI_K3_GEMM_AR`; `vllm/models/kimi_k3/nvidia/model.py` 把 `maybe_init_gemm_rs` 重构为 `maybe_init_gemm_rs_ar`, 以 `all_reduce = not use_sequence_parallel` 决定模式 (SP 开启走 RS、关闭走 AR), 保留原有 BF16/SM100 系 /TP 2-16 且 128 可被 TP 整除的校验链; `init_gemm_rs_ar` 初始化时若 `multicast_ptr == 0` (典型如跨节点无 NVLink 多播) 会

抛出 `RuntimeError`，由上层 `try-except` 捕获后告警并降级禁用，回退到普通 `o_proj` 线性层。

3. 模型调用链接入。`KimiMLP`、`KimiKda`、`KimiMla` 从布尔标志 `run_gemm_rs` 切换到 `self.gemm_rs_ar` 单例引用：初始化时用 `get_gemm_rs_ar().can_run(weight)` 判断投影形状是否兼容，forward 时用 `should_run(input)` 判断 token 数是否达到 $M \geq 128$ 的调度门槛，命中即返回融合内核结果，否则落入常规 `o_proj`。注释明确 RS 要求 `sequence sharding`，而 AR 作用于 `replicated tokens`。
4. 测试、基准、预热与 CI 配套。单测与 benchmark 整体重命名为 `*_gemm_rs_ar.py`：单测抽出 `_run_mode` 双模式参数化，`_reference` 在 AR 模式下改用 `dist.all_reduce` 作参考实现，并新增 AR 专属的 `workspace` 复用生命周期校验（首次输出快照在二次调用后仍须有效）；benchmark 新增 `--mode rslar` 参数与 `make_torch_ring_ll_gemm_collective`、`make_torch_ldmc_gemm_collective`、`make_fused_gemm_collective` 等集体通信基线；`vllm/model_executor/warmup/kernel_warmup.py` 新增 `_warmup_kimi_k3_gemm_rs_ar`，用 `sys.modules.get` 探测 `gemm_rs_ar` 模块是否已加载，避免非 Kimi-K3 模型触发无关 JIT 编译；`.buildkite/test_areas/distributed.yaml` 同步更新分布式测试覆盖范围。

关键文件：

- `vllm/models/kimi_k3/nvidia/ops/cute_dsl/gemm_rs_ar.py`（模块 融合内核；类别 `source`；类型 `core-logic`；符号 `multimem_st_16B`, `Sm100GemmRsArBF16`, `GemmRsAr`, `init_gemm_rs_ar`）：核心内核文件：新增 `multimem_st_16B` 多播存储 DSL 算子，`Sm100GemmRsBF16` 泛化为 `Sm100GemmRsArBF16` 并支持 `all_reduce` 模式，是 GEMM-AR 的实现主体
- `vllm/models/kimi_k3/nvidia/model.py`（模块 模型集成；类别 `source`；类型 `integration`；符号 `maybe_init_gemm_rs_ar`, `maybe_init_gemm_rs`, `KimiMLP`）：模型集成入口：`maybe_init_gemm_rs` 重构为 `maybe_init_gemm_rs_ar`，按 SP 拓扑选择 RS/AR 单例，并加入 NVLink 多播缺失时的优雅降级
- `tests/kernels/test_kimi_k3_gemm_rs_ar.py`（模块 内核测试；类别 `test`；类型 `test-coverage`；符号 `_run_mode`, `_reference`, `_assert_valid_rows_close`, `test_kimi_k3_gemm_rs_ar`）：单测从 RS-only 扩展为 RS/AR 双模式，并新增 AR 专属的 `workspace` 复用生命周期校验，直接锚定本 PR 的核心限制
- `vllm/models/kimi_k3/nvidia/kda.py`（模块 注意力模块；类别 `source`；类型 `integration`；符号 `KimiKda`, `KimiAttention`）：注意力 O-proj 调用链从 `run_gemm_rs` 布尔标志切换到 `gemm_rs_ar` 单例，forward 中通过 `should_run` 决定是否走融合内核
- `vllm/model_executor/warmup/kernel_warmup.py`（模块 预热逻辑；类别 `source`；类型 `integration`；符号 `_warmup_kimi_k3_gemm_rs_ar`, `kernel_warmup`）：新增 `_warmup_kimi_k3_gemm_rs_ar` 预热钩子，用 `sys.modules.get` 探测模块加载状态，避免非 Kimi-K3 模型引入 JIT 编译开销
- `benchmarks/kernels/benchmark_kimi_k3_gemm_rs_ar.py`（模块 性能基准；类别 `source`；类型 `benchmark`；符号 `make_torch_ring_ll_gemm_collective`, `make_torch_ldmc_gemm_collective`, `make_fused_gemm_collective`, `make_ring_ll_collective`）：benchmark 整体重命名并新增 `--mode rslar`，补充环形 / 树形 / torch+fused 集体通信基线，用于量化 AR 相对 torch.mm+NCCL all-reduce 的收益

- vllm/models/kimi_k3/nvidia/mla.py (模块 注意力模块; 类别 source; 类型 integration; 符号 KimiMla) : MLA 模块与 kda.py 同步切换 gemm_rs_ar 单例, 保持注意力输出投影路径行为一致
- vllm/envs.py (模块 环境变量; 类别 source; 类型 configuration; 符号 VLLM_KIMI_K3_GEMM_AR) : 新增 VLLM_KIMI_K3_GEMM_AR 环境变量, 与既有 VLLM_KIMI_K3_GEMM_RS 并列控制 AR 模式开关
- .buildkite/test_areas/distributed.yaml (模块 CI 配置; 类别 config; 类型 configuration) : 分布式测试区域配置随测试文件重命名与双模式扩展同步更新, 保证 CI 覆盖

关键符号: multimem_st_16B, Sm100GemmRsArBF16.init, Sm100GemmRsArBF16.kernel, init_gemm_rs_ar, get_gemm_rs_ar, warmup_gemm_rs_ar, maybe_init_gemm_rs_ar, KimiMLP.init, KimiKda.forward, KimiAttention.forward, _warmup_kimi_k3_gemm_rs_ar, _run_mode, benchmark_shape

关键源码片段

vllm/models/kimi_k3/nvidia/ops/cute_dsl/gemm_rs_ar.py

核心内核文件: 新增 multimem_st_16B 多播存储 DSL 算子, Sm100GemmRsBF16 泛化为 Sm100GemmRsArBF16 并支持 all_reduce 模式, 是 GEMM-AR 的实现主体

```
# ===== multimem_st_16B: 多播存储原语 =====
# 将 16B 向量写入 NVLink 多播地址, 使同一 NVLink 域内所有 rank 都读到同一份数据。
# 相比 GEMM-RS 的单播 store, 这是 GEMM-AR 能实现内核内 all-reduce 的关键。
@dsl_user_op
def multimem_st_16B(dst: cute.Tensor, value: cute.Tensor, *, loc=None, ip=None) -> None:
    # 向量指令要求 dst 与 value 是连续的 BF16x8 向量, 且 dst 16 字节对齐。
    assert dst.element_type == BFloat16
    assert value.element_type == BFloat16
    ptr = dst.iterator.toint(loc=loc, ip=ip).ir_value(loc=loc, ip=ip)
    regs = cute.recast_tensor(value, Int32, loc=loc, ip=ip)
    # 内联汇编发射 PTX: multimem.st.relaxed.sys.global.v4.f32 一次写 4 个寄存器。
    llvm.inline_asm(
        None,
        [ptr] + [regs[i].ir_value(loc=loc, ip=ip) for i in range(4)],
        "multimem.st.relaxed.sys.global.v4.f32 [$0], {$1, $2, $3, $4};",
        "l,r,r,r,r",
        has_side_effects=True,
        loc=loc,
        ip=ip,
    )

# ===== Sm100GemmRsArBF16: RS/AR 双模式内核 =====
class Sm100GemmRsArBF16:
    """SM100 BF16 GEMM, 内核内融合 TP 维 reduce-scatter (RS) 或 all-reduce (AR) 。”"""

    def __init__(
```

```

self,
*,
rank: int,
num_ranks: int,
BN: int = 128,
cta_group: int = 1,
all_reduce: bool = False,
) -> None:
    self.rank = rank
    self.num_ranks = num_ranks
    BM, BK = 128, 64
    self.cta_tile = (BM, BN, BK)
    self.cta_group = cta_group
    self.all_reduce = all_reduce # 决定写回路径与 local_M 语义

# ---- kernel() 内与 RS/AR 相关的核心分支 (节选) ----
# AR 模式下每个 rank 持有全局 partial 的部分和, local_M 取总行数再按 rank 数
# 切分; RS 模式下 output 只是本 rank 的序列分片, local_M 直接取 output 行数。
local_M = (
    partial_uc.shape[0] // num_ranks
    if self.all_reduce
    else output.shape[0]
)

# 通信 warp (tid 4-7 区间) 负责跨 rank 归约; epilogue 仅在 RS 模式需要把
# 结果切分写入 output, AR 模式由 multmem_st_16B 直接多播写回 workspace。
if cutlass.const_expr(not self.all_reduce):
    output_vecs = cute.zipped_divide(output, (1, vec_width))

```

vllm/models/kimi_k3/nvidia/model.py

模型集成入口: maybe_init_gemm_rs 重构为 maybe_init_gemm_rs_ar, 按 SP 拓扑选择 RS/AR 单例, 并加入 NVLink 多播缺失时的优雅降级

```

def maybe_init_gemm_rs_ar(vllm_config: VllmConfig, use_sequence_parallel: bool) -> bool:
    # 两个特性开关可能同时打开, 但 worker 的静态 SP 拓扑决定单例只能绑定一种
    # 模式: 开启 sequence parallel 时运行 GEMM-RS, 否则运行 GEMM-AR。
    all_reduce = not use_sequence_parallel
    mode = "GEMM-AR" if all_reduce else "GEMM-RS"
    enabled = envs.VLLM_KIMI_K3_GEMM_AR if all_reduce else envs.VLLM_KIMI_K3_GEMM_RS
    if not enabled:
        return False

    parallel_config = vllm_config.parallel_config
    tp_size = parallel_config.tensor_parallel_size
    # 逐项校验运行条件, 任一不满足就给出禁用原因并回退到普通 GEMM 路径。
    if parallel_config.use_ubatching:
        reason = "ubatching is enabled"
    elif vllm_config.model_config.dtype != torch.bfloat16:
        reason = "the model dtype is not BF16"

```

```

elif not current_platform.is_cuda():
    reason = "the device is not CUDA"
elif not current_platform.is_device_capability_family(100):
    reason = "the device is not SM100-family"
elif not 1 < tp_size <= 16:
    reason = "TP size is not in the supported range 2-16"
elif 128 % tp_size != 0:
    reason = "TP size does not divide 128"
else:
    reason = None

if reason is not None:
    logger.warning_once("%s is disabled because %s.", mode, reason)
    return False

from vllm.models.kimi_k3.nvidia.ops.cute_dsl.gemm_rs_ar import init_gemm_rs_ar

config = vllm_config.model_config.hf_text_config
try:
    init_gemm_rs_ar(
        max_M=vllm_config.scheduler_config.max_num_batched_tokens,
        N=config.hidden_size,
        all_reduce=all_reduce,
    )
except RuntimeError as e:
    # 典型场景：TP rank 不在同一 NVLink 域（如跨 2 个 B200 节点），
    # 对称内存多播指针为 0；捕获后告警并降级，保证服务仍可用。
    logger.warning_once(
        "%s is disabled because initialization failed: %s. This may mean "
        "the TP ranks do not share one NVLink domain.",
        mode,
        e,
    )
    return False
if all_reduce:
    logger.info_once(
        "GEMM-AR is enabled. To disable it, set VLLM_KIMI_K3_GEMM_AR=0."
    )
else:
    logger.info_once("GEMM-RS is enabled.")
return True

```

tests/kernels/test_kimi_k3_gemm_rs_ar.py

单测从 RS-only 扩展为 RS/AR 双模式，并新增 AR 专属的 workspace 复用生命周期校验，直接锚定本 PR 的核心限制

```

def _reference(
    x: torch.Tensor,
    weight: torch.Tensor,

```

```

world_size: int,
group: dist.ProcessGroup,
all_reduce: bool,
) -> torch.Tensor:
    # 参考实现: torch.mm 计算全局 partial, 再按模式补一次集体通信。
    M = x.shape[0]
    padded_M = (M + world_size - 1) // world_size * world_size
    partial = torch.empty(
        (padded_M, weight.shape[0]),
        dtype=x.dtype,
        device=x.device,
    )
    torch.mm(x, weight.T, out=partial[:M])
    if padded_M > M:
        partial[M:].zero_()

    if all_reduce:
        # AR 语义: 所有 rank 拿到完整的全局结果。
        dist.all_reduce(partial, group=group)
        return partial[:M]

    # RS 语义: reduce-scatter 到每个 rank 的序列分片。
    output = torch.empty(
        (padded_M // world_size, weight.shape[0]),
        dtype=x.dtype,
        device=x.device,
    )
    dist.reduce_scatter_single(output, partial, group=group)
    return output

def _run_mode(
    *,
    all_reduce: bool,
    device: torch.device,
    group: dist.ProcessGroup,
    rank: int,
    world_size: int,
    weights: dict[int, torch.Tensor],
) -> None:
    # cute_dsl 仅在 CUDA 上可用, 所以在 GPU worker 内部再导入, 避免模块级导入失败。
    from vllm.models.kimi_k3.nvidia.ops.cute_dsl.gemm_rs_ar import GemmRsAr

    gemm_rs_ar = GemmRsAr(
        max_M=max(M for M, _ in _SHAPES),
        N=_N,
        all_reduce=all_reduce,
    )
    input_generator = torch.Generator(device=device)

```

```

# 交替形状覆盖不同 grid 大小、CTA-group 选择与 BN=128/BN=256 分派。
for M, K in (*_SHAPES, *_SHAPES[:::-1]):
    input_generator.manual_seed(2000 + M + K)
    x = torch.randn(
        M,
        K,
        dtype=torch.bfloat16,
        device=device,
        generator=input_generator,
    )
    expected = _reference(x, weights[K], world_size, group, all_reduce)
    actual = gemm_rs_ar(x, weights[K])
    torch.accelerator.synchronize(device)
    _assert_valid_rows_close(actual, expected, M, rank, all_reduce)

if all_reduce:
    # AR 输出落在对称内存多播 workspace 上，第二次调用会复用同一块内存；
    # 这里验证 workspace 复用后第一次的结果快照仍有效（对应 PR body 中
    # 提到的输出生命周期限制）。
    lifetime_M, lifetime_K = 257, 768
    input_generator.manual_seed(2500)
    lifetime_x = torch.randn(
        lifetime_M,
        lifetime_K,
        dtype=torch.bfloat16,
        device=device,
        generator=input_generator,
    )
    first_output = gemm_rs_ar(lifetime_x, weights[lifetime_K])
    first_snapshot = first_output.clone()
    second_output = gemm_rs_ar(-lifetime_x, weights[lifetime_K])
    # 后续断言：first_snapshot 必须仍与 first_output 一致，second_output
    # 必须与基于 -lifetime_x 的参考 all-reduce 结果一致。

```

评论区精华

核心讨论集中在 `gemm_rs_ar.py` 第 731 行的 NVLink 多播校验：wzhao18 质疑默认启用融合时，遇到无 NVLink 多播的环境（例如 2 个 B200 节点组成的 TP）不应直接 raise，建议返回 `None` 并记录不支持日志；随后他自查确认 `model.py` 的 `maybe_init_gemm_rs_ar` 已用 try-except 包裹 `init_gemm_rs_ar`，捕获 `RuntimeError` 后告警并降级禁用，于是撤回意见（"never mind just saw the try-except handling. so it is safe."），并 APPROVE 且请 zhyongye 二次复核后合入。另外由于 PR 来自 fork，Claude 自动 review 被跳过（`claude[bot]` 注明 `automated review is disabled`），只能依赖人工审核。

- 无 NVLink 多播环境下初始化失败的处理方式 (design): wzhao18 随即自我确认 `model.py` 的 `maybe_init_gemm_rs_ar` 已用 try-except 包裹 `init_gemm_rs_ar`，`RuntimeError` 会被捕获并告警降级，因此保留 raise 是安全的（"never mind just saw the try-except handling. so it is safe."）。

- fork PR 自动 review 与双人复核流程 (other): zhyongye 复核后 APPROVE 并合并，审查流程闭环。

风险与影响

- 风险:

1. 多播 workspace 复用竞态 (核心风险) : multimem.st 写入的是初始化时分配的对称内存多播 workspace, 而非调用方新分配缓冲区; 若消费者未读完就再次调用 GEMM-AR, 旧结果会被覆盖。PR 通过 $M \geq 128$ 才启用分派 (小 batch decode 不经过该路径) 和测试中的生命周期校验来缓解, 但根本消除依赖未来把输出 clone 融合进 kernel (PR body 已列为 future work) 。
2. NVLink 多播依赖: 跨节点 TP (如 2 个 B200 节点) 没有多播指针时 `init_gemm_rs_ar` 会 raise, 目前依赖 `maybe_init_gemm_rs_ar` 的 try-except 降级; 若未来有其他调用点未包裹该异常, 可能导致启动失败。review 中 wzhaol8 曾就这一点提出疑虑。
3. 文件重命名连锁影响: `gemm_rs.py` -> `gemm_rs_ar.py`、single test 和 benchmark 同步改名, 任何仍引用旧路径的代码会漂移; 例如历史 PR #53113 修改过旧测试文件 `tests/kernels/test_kimi_k3_gemm_rs.py` 的 ROCm 跳过名单, 需要确认 CI 侧引用已同步 (本 PR 已更新 `.buildkite/test_areas/distributed.yaml`) 。
4. 对时序敏感的 warp 分工: kernel 注释明确 "Swapping the warpgroups can hang due to warp scheduling", epilogue 与通信 warp 的划分对调度敏感, 改动底层调度器时需回归此内核。
5. warmup 依赖模块加载状态: `_warmup_kimi_k3_gemm_rs_ar` 用 `sys.modules.get` 探测模块是否加载, 若模型构造顺序变化导致模块尚未加载, 预热会被跳过, 首次执行可能触发 JIT 编译延迟。- 影响: 影响范围限定在 Kimi-K3 模型 + SM100 系硬件 (GB300 等) + TP>1 的部署: SP 开启时走原有 GEMM-RS, SP 关闭时新启用 GEMM-AR 路径。E2E 收益以延迟为主——并发 1-32 下 TTFT 降低 2.84%-5.74%, 吞吐变化在 -0.5%~+1.2% 之间; 微基准大 M 收益显著 (M=32768 时 1.21x), 小 M (M<128) 反而变慢, 因此设计了 $M \geq 128$ 调度门槛。新增 `VLLM_KIMI_K3_GEMM_AR` 环境变量用于开关。对团队而言, 内核文件、测试与 benchmark 的统一重命名使 Kimi-K3 内核演进保持单一入口, 也为后续消除输出 clone 的工作预留了测试锚点; 普通 CUDA/ROCm 环境因 SM100、BF16、TP 多重校验不受影响。- 风险标记: 多播 workspace 复用存在时序竞态, 依赖 NVLink 多播能力, 缺失时依赖降级路径, 内核文件重命名波及下游 CI 引用, 限定 SM100/BF16/TP 组合, AR 分派按 $M \geq 128$ 门控

关联脉络

- PR #53113 [CI/Build][ROCm] Keep the CUDA-only kernel tests out of the ROCm run: 该 PR 修改过 `tests/kernels/test_kimi_k3_gemm_rs.py` 的 ROCm 跳过名单, 而本 PR 将该文件重命名为 `test_kimi_k3_gemm_rs_ar.py`, 需要 CI 引用同步, 避免跳过逻辑漂移。
- PR #53040 [DSV4][Kernel] Fuse shared experts into MegaMoE: 同期也改动 `vllm/models/kimi_k3/nvidia/model.py`, 说明该模型目录处于活跃演进期, 合入时需注意同一文件上的并发修改冲突。