PR #33905 完整报告

sgl-project/sglang

[XPU] Pad MoE expert weight row stride to avoid L3 aliasing

合并时间：2026-08-12 06:54

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/33905

执行摘要

- 一句话：XPU MoE 权重行 stride 填充 64B 规避 L3 aliasing
- 推荐动作：值得精读。该 PR 把硬件内存系统（L3 set aliasing）分析转化成一段很小但精准的分配逻辑，展示了 ' 不改逻辑 shape、只调 stride' 的兼容性设计，并配套了设备门控与 bit-exact 数值测试。对关注 Intel XPU 性能或 MoE kernel 调优的工程师尤其有参考价值；后续在 XPU 上做其他 GEMM 算子优化时可复用相同的 padding 常量与判定函数。

功能与动机

PR body 说明动机是 'Padding the MoE weight for specific shapes on XPU'，即 XPU 上特定形状的 MoE 权重需要 padding。首个 commit message 给出更详细背景：sgl-kernel-xpu 的 Xe20 grouped GEMM 逐行遍历 B 操作数，行的字节大小决定其在 L3 中的 set 分布；当行字节数是 2048 的倍数且奇数因子 >= 3（bf16 下对应 K = 3072、7168 等）时，连续行会折叠到少数几个 L3 set 造成 thrash，拖慢权重装载。sgl-kernel-xpu bd7ab1b 已让 kernel 从张量读取行 stride，本 PR 在 SGLang 侧补上 stride padding 的配套改动。

实现拆解

实现按以下 4 步拆解：

1. L3 aliasing 判定（python/sglang/srt/layers/moe/utils.py）：新增常量 XPU_MOE_LD_PADDING_BYTES = 64 与函数 xpu_moe_ld_padding_elems(k_dim, itemsize)。判定规则基于行字节数 row_bytes = k_dim * itemsize：先取 row_bytes 的 2 因子数量 trailing_zeros，再算奇数因子 odd_cofactor；当 trailing_zeros >= 11（行字节数是 2048 的倍数）且 odd_cofactor >= 3 时返回 64 // itemsize 个元素作为 padding，否则返回 0。该判定以字节为单位，天然适配 bf16/fp8/int8 等不同 dtype。
2. XPU 门控（python/sglang/srt/layers/quantization/unquant.py）：新增 _use_xpu_moe_ld_padding(use_triton_kernels)，要求同时满足 use_intel_xpu_backend()、torch.get_default_device().type == "xpu"、且非 Triton 路径。前两个条件避免在 XPU 存在于机器上但权重实际构建于 CPU/CUDA 时误填充；排除 Triton 路径是因为 Triton 侧 B 矩阵转置存储、不读取行 stride。
3. 填充分配（unquant.py）：新增 _empty_xpu_moe_expert_weight(num_experts, n_dim, k_dim, dtype)，当判定需要 padding 时分配 [E, N, K + pad] 的 buffer 并 narrow 成 [E, N, K] 的视图返回；逻辑 shape 与普通 torch.empty 完全一致，因此权重加载器按 shape

索引的代码无需改动。在 UnquantizedFusedMoEMethod.create_weights 中，w13_weight 与 w2_weight 两条分配路径都按 pad_ld_for_xpu 分支选择 padded 或普通分配。

4. 测试配套（test/registered/xpu/test_moe_ld_padding.py）：新增 180 行测试，覆盖 padding 形状选择（3072/5120/6144/7168/14336 等触发、1024/2048/4096 等不触发）、itemsize 缩放、分配后 shape 不变且 stride(1) == K + pad、设备门控（CPU 环境下保持 contiguous、关掉 backend 后不填充）、loader 风格 narrow + copy_ 语义保真，以及 TestXpuMoePaddedWeightsNumerics 用 fused_experts 验证 padded 与普通权重输出 bit-identical。测试注册到 XPU CI 套件 stage-b-test-1-gpu-xpu。

关键文件：

- python/sglang/srt/layers/quantization/unquant.py（模块 量化层；类别 source；类型 core-logic；符号 _use_xpu_moe_ld_padding, _empty_xpu_moe_expert_weight, UnquantizedFusedMoEMethod.create_weights）：核心改动文件：新增 _use_xpu_moe_ld_padding 门控与 _empty_xpu_moe_expert_weight 填充分配函数，并修改 UnquantizedFusedMoEMethod.create_weights 让 w13/w2 两条权重分配路径按设备上下文选择 padded 分配。
- python/sglang/srt/layers/moe/utils.py（模块 MoE 工具；类别 source；类型 core-logic；符号 xpu_moe_ld_padding_elems, XPU_MOE_LD_PADDING_BYTES）：新增 XPU_MOE_LD_PADDING_BYTES 常量与 xpu_moe_ld_padding_elems 判定函数，集中表达 L3 aliasing 判定规则与 padding 字节数权衡，是本次优化的算法核心。
- test/registered/xpu/test_moe_ld_padding.py（模块 XPU 测试；类别 test；类型 test-coverage；符号 TestXpuMoeLdPadding, test_padding_selects_aliasing_shapes, test_padding_scales_with_itemsize, test_allocation_keeps_shape_and_pads_stride）：新增测试文件，覆盖 padding 判定、分配 shape/stride、设备门控、loader 风格 copy 语义以及 padded 权重与普通权重的 bit-exact 数值一致性，是本改动的回归保障。

关键符号：xpu_moe_ld_padding_elems, _use_xpu_moe_ld_padding, _empty_xpu_moe_expert_weight, UnquantizedFusedMoEMethod.create_weights

关键源码片段

python/sglang/srt/layers/quantization/unquant.py

核心改动文件：新增 _use_xpu_moe_ld_padding 门控与 _empty_xpu_moe_expert_weight 填充分配函数，并修改 UnquantizedFusedMoEMethod.create_weights 让 w13/w2 两条权重分配路径按设备上下文选择 padded 分配。

```
def _use_xpu_moe_ld_padding(use_triton_kernels: bool) -> bool:
    # use_intel_xpu_backend() 只能说明机器上有 XPU，不代表当前权重构建在
    # XPU 上（环境变量可能在 CPU/CUDA 服务场景下也被设置）。create_weights
    # 不接收 device 参数，而是通过 torch.device 上下文决定分配位置，所以这里
    # 必须同时检查默认设备类型，避免把非 XPU 权重 pad 成非连续张量。
    return (
        use_intel_xpu_backend()
        and torch.get_default_device().type == "xpu"
```

```
        and not use_triton_kernels
    )


def _empty_xpu_moe_expert_weight(num_experts, n_dim, k_dim, dtype):
    pad = xpu_moe_ld_padding_elems(k_dim, dtype.itemsize)
    if pad == 0:
        return torch.empty(num_experts, n_dim, k_dim, dtype=dtype)
    # 多分配 K + pad，再用 narrow 视图切回 K，逻辑 shape 不变（权重加载器
    # 只按 shape 索引，因此无需改动），但每行的实际 stride 变成 K + pad，
    # 从而打破 L3 set aliasing。返回的视图非连续，只有 K 切片会被读写。
    return torch.empty(num_experts, n_dim, k_dim + pad, dtype=dtype)[:, :, :k_dim]

# create_weights 内部的分配分支（节选）
pad_ld_for_xpu = _use_xpu_moe_ld_padding(self.use_triton_kernels)

# Fused gate_up_proj（column parallel）
w13_up_dim = (
    2 * intermediate_size_per_partition
    if layer.moe_runner_config.is_gated
    else intermediate_size_per_partition
)
w13_weight_n, w13_weight_k = (w13_up_dim, hidden_size)
if self.use_triton_kernels:
    w13_weight_n, w13_weight_k = w13_weight_k, w13_weight_n
if pad_ld_for_xpu:
    w13_weight_data = _empty_xpu_moe_expert_weight(
        num_experts, w13_weight_n, w13_weight_k, params_dtype
    )
else:
    w13_weight_data = torch.empty(
        num_experts, w13_weight_n, w13_weight_k, dtype=params_dtype
    )
w13_weight = torch.nn.Parameter(w13_weight_data, requires_grad=False)
layer.register_parameter("w13_weight", w13_weight)
```

python/sglang/srt/layers/moe/utils.py

新增 XPU_MOE_LD_PADDING_BYTES 常量与 xpu_moe_ld_padding_elems 判定函数，集中表达 L3 aliasing 判定规则与 padding 字节数权衡，是本次优化的算法核心。

```
# 行 stride 填充量，单位字节。64B 与 sgl-kernel-xpu MoE benchmark 使用的
# 32 个 bf16 元素一致；注释记录 BMG 上的实测：32B 仍可消除 aliasing 但
# hidden=7168 时反而比不 pad 慢约 6%，128B 相对 64B 无收益。
XPU_MOE_LD_PADDING_BYTES = 64


def xpu_moe_ld_padding_elems(k_dim: int, itemsize: int) -> int:
    # Xe20 grouped GEMM 逐行遍历 B，行字节数决定行落在哪个 L3 set（地址位
    # XOR 折叠）。当行字节数是 2048 的倍数且奇数因子 >= 3（bf16 下 K =
```

```
    # 3072、7168 等）时，连续行会折叠到少数 set 上造成 thrash。
    row_bytes = k_dim * itemsize
    if row_bytes <= 0 or XPU_MOE_LD_PADDING_BYTES % itemsize != 0:
        return 0
    trailing_zeros = (row_bytes & -row_bytes).bit_length() - 1
    odd_cofactor = row_bytes >> trailing_zeros
    if trailing_zeros >= 11 and odd_cofactor >= 3:
        return XPU_MOE_LD_PADDING_BYTES // itemsize
    return 0
```

评论区精华

PR 没有技术性 review 评论，comments 均为流程交互：作者 @mingfeima 请求 Intel 侧评审；中途触发 '/rerun-failed-ci' 重跑 CI；合并前作者向 @alexnails 说明 'the CI failures not related to this PR, can you merge it?'。值得注意的隐含讨论点是：CI 曾出现与本 PR 无关的失败，最终由维护者确认后合并，说明该改动本身的风险面较小。

- 暂无高价值评论线程

风险与影响

- 风险：具体风险包括：
  1. 非连续权重视图：padded 权重 is_contiguous() == False，若未来有其他后端或算子路径直接假设权重连续（如 deep_gemm、flashinfer cutlass 等），可能在 XPU 上出问题；当前只对非 Triton 的 XPU 路径生效，且测试验证了 loader 的 copy_ 语义。
  2. 设备上下文依赖：_use_xpu_moe_ld_padding 依赖 torch.get_default_device()，若模型加载流程在某个尚未切换默认设备的阶段调用 create_weights，会出现该填充而未填充（性能回退）或不该填充而填充（非连续）的情况。测试通过 mock 覆盖了 CPU/XPU 两种上下文。
  3. kernel 版本耦合：填充生效的前提是 sgl-kernel-xpu 侧（PR#187 / bd7ab1b）支持从张量读取行 stride；若用户安装的 sgl-kernel-xpu 版本回退或未包含该改动，padded stride 会被 kernel 忽略而可能算错。
  4. 内存开销：每个触发 K 的专家权重多分配 64 * E * N 字节；对 32 专家、N=7168 的 w13 场景约多 14 MB 级别，影响很小但需知晓。
  5. 测试覆盖：数值 bit-exact 测试依赖 torch.xpu.is_available()，非 XPU CI 上会 skip。
- 影响：影响范围集中在 XPU 平台的未量化 MoE 权重加载路径：
  - 对 XPU 用户：K 落在 aliasing 区间的模型（如 bf16 下 K=3072/7168 的 MoE 模型）在 grouped GEMM 上有望避免 L3 set thrash，提升 prefill/decode 权重装载性能；逻辑 shape 不变，加载器、调度器的行为均无感知。
  - 对非 XPU 用户（CUDA/CPU/NPU/AMD）：_use_xpu_moe_ld_padding 在非 XPU 默认设备下恒为 False，分配路径与之前完全一致，零行为变化。
  - 对团队：新增一条 XPU 专属的分配分支与常量，需要在 sgl-kernel-xpu 侧保持同步演进；测试挂入 XPU CI stage-b，增强回归保障。

- 风险标记：XPU 专属路径，非连续权重视图，依赖 sgl-kernel-xpu 版本，设备上下文敏感

关联脉络

- PR #34795 [MoE] Add H20 fp8_w8a8 tuned configs for Qwen3.8 (triton 3.7.1) + fix Qwen3_5MoeForCausalLM tuning: 同为 MoE 执行路径的硬件特化性能调优，但平台不同（H20 vs XPU），说明 MoE 性能调优持续按平台展开。
- PR #35020 [Fix] Correct dense FP8 Marlin bias ordering: 同属权重量化 / 布局修正类改动，但与 XPU padding 无直接依赖，关联较弱。