PR #21668 完整报告

sgl-project/sglang

[XPU] Enable qwen3.5 on XPU

合并时间：2026-05-18 14:59

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/21668

执行摘要

- 一句话：在 Intel GPU (XPU) 上支持 Qwen3.5 模型
- 推荐动作：建议仔细阅读 chunk_delta_h.py 与 chunk_fwd.py 中的低寄存器设计模式，以及 is_intel 条件导入的组织方式。该 PR 为后续其他非 CUDA 硬件支持提供了参考架构（vendor 目录 + 工具函数抽象）。对 Intel GPU 推理性能感兴趣的工程师亦可关注。

功能与动机

PR body 明确目标是 Enable Qwen3.5 series of models on XPU (Intel GPU)。review 中进一步指出，现有 triton kernel 不适用于非 CUDA GPU arch，Intel GPU 的限制（prefer tensor descriptor, limited registers）要求针对性调整 kernel。

实现拆解

1. 新增 XPU 优化的 FLA kernel：在 python/sglang/srt/hardware_backend/xpu/kernels/fla/ 下创建 chunk_delta_h.py、chunk_fwd.py 和 fused_sigmoid_gating_recurrent.py。这些 kernel 使用 make_tensor_descriptor 替代 tl.make_block_ptr 以规避 triton-xpu 的 bug，并通过显式 K 循环（chunk_delta_h.py）和低寄存器 KKT 解法（chunk_fwd.py）控制寄存器压力，适配 Intel GPU 架构。
2. 添加 MROPE XPU 前向传播：在 python/sglang/srt/layers/rotary_embedding/mrope.py 中新增 forward_xpu 方法，根据 positions 维度选择调用 forward_triton 或 forward_native，使 Qwen3.5 的多模态旋转位置编码能在 XPU 上正确计算。
3. 条件导入与 is_intel 分支：在 python/sglang/srt/layers/attention/fla/chunk.py、kda.py、gdn_triton.py 等文件中，导入 is_intel 标志，当为真时使用 XPU 专用 kernel。同时将公共的 BV 上限抽象为 get_block_size_v_cap() 工具函数，避免硬编码。
4. 性能分析与 CI 支持：修改 _save_profile_trace_results（bench_one_batch.py）增加对 XPU 设备的排序字段（self_xpu_time_total）。更新 ci_register.py 注册 XPU 测试套件，调整 test_chunk_gated_delta_rule.py 自动检测设备。
5. 安装与文档更新：更新安装文档和 Dockerfile，指定使用修复了 block_ptr bug 的 triton-xpu 版本。

关键文件：

- python/sglang/srt/hardware_backend/xpu/kernels/fla/chunk_delta_h.py（模块 分块注意力；类别 source；类型 core-logic；符号 chunk_gated_delta_rule_fwd_kernel_h_blockdi

m64_k_loop, chunk_gated_delta_rule_fwd_h, grid）：新增 XPU 专用的 chunk_gated_delta_rule_fwd_h kernel，使用 tensor descriptor 和 K 循环减少寄存器压力，是性能改进的核心。

- python/sglang/srt/hardware_backend/xpu/kernels/fla/chunk_fwd.py（模块 分块注意力；类别 source；类型 core-logic；符号 chunk_gated_delta_rule_fwd_kkt_solve_kernel_low_reg, chunk_gated_delta_rule_fwd_intra）：新增 XPU 专用的 chunk_gated_delta_rule_fwd_intra kernel，采用低寄存器 KKT 解法，分两遍计算对角和非对角块。
- python/sglang/srt/hardware_backend/xpu/kernels/fla/fused_sigmoid_gating_recurrent.py（模块 分块注意力；类别 source；类型 core-logic；符号 fused_sigmoid_gating_delta_rule_update）：新增 XPU 版本 fused_sigmoid_gating_delta_rule_update，调用 triton kernel 时设置 num_warps=1 和受限 BV 以减少寄存器压力。
- python/sglang/srt/layers/rotary_embedding/mrope.py（模块 旋转编码；类别 source；类型 core-logic；符号 forward_xpu）：添加 forward_xpu 方法，支持 MROPE 位置编码在 XPU 上的前向传播。
- python/sglang/srt/layers/attention/fla/chunk.py（模块 注意力层；类别 source；类型 dependency-wiring）：通过 is_intel 条件导入 XPU 专用 kernel，替换默认 CUDA 实现。
- python/sglang/bench_one_batch.py（模块 性能分析；类别 source；类型 core-logic；符号 _save_profile_trace_results）：支持 XPU 设备类型的 profile 结果排序。
- python/sglang/srt/layers/attention/fla/kda.py（模块 注意力层；类别 source；类型 dependency-wiring）：添加 is_intel 条件导入 XPU 版 kernel。
- python/sglang/srt/layers/attention/linear/kernels/gdn_triton.py（模块 注意力层；类别 source；类型 dependency-wiring）：添加 is_intel 条件导入 XPU 版 kernel。
- python/sglang/test/ci/ci_register.py（模块 CI 注册；类别 test；类型 test-coverage；符号 register_xpu_ci）：注册 XPU CI 测试，确保 XPU 相关测试在 CI 中运行。
- test/registered/attention/test_chunk_gated_delta_rule.py（模块 单元测试；类别 test；类型 test-coverage）：调整测试以支持 XPU 设备，包括设备检测和 CI 注册。
- python/sglang/srt/layers/attention/fla/layernorm_gated.py（模块 注意力层；类别 source；类型 core-logic）：添加 is_intel 条件分支以兼容 XPU。
- python/sglang/srt/hardware_backend/xpu/__init__.py（模块 XPU 初始化；类别 source；类型 entrypoint）：标记 XPU 后端模块。

关键符号：chunk_gated_delta_rule_fwd_h, chunk_gated_delta_rule_fwd_intra, chunk_gated_delta_rule_fwd_kkt_solve_kernel_low_reg, fused_sigmoid_gating_delta_rule_update, forward_xpu, _save_profile_trace_results, register_xpu_ci

关键源码片段

python/sglang/srt/hardware_backend/xpu/kernels/fla/chunk_delta_h.py

新增 XPU 专用的 chunk_gated_delta_rule_fwd_h kernel，使用 tensor descriptor 和 K 循环减少寄存器压力，是性能改进的核心。

```
# 在 kernel 内部，使用 tensor descriptor 而非 block_ptr
# 以绕过 triton-xpu 早期版本中 block_ptr 的边界检查问题
w_desc = make_tensor_descriptor(
    base=w, shape=(T, K), strides=(stride_w, 1), block_shape=(BT, 64),
)
v_desc = make_tensor_descriptor(
    base=v, shape=(T, V), strides=(stride_v, 1), block_shape=(BT, BV),
)
k_desc = make_tensor_descriptor(
    base=k, shape=(T, K), strides=(stride_k, 1), block_shape=(BT, 64),
)
# 主递归：显式循环 K 维度，每次处理 64 个通道
for k_start in range(0, K, 64):
    b_h1 = tl.zeros([BV, 64], dtype=tl.float32)
    # 加载初始状态（如果启用）
    if USE_INITIAL_STATE:
        p_h0_1 = tl.make_block_ptr(
            h0, (V, K), (K, 1), (i_v * BV, k_start), (BV, 64), (1, 0)
        )
        b_h1 += tl.load(p_h0_1, boundary_check=(0, 1)).to(tl.float32)
    # 在时间步上迭代
    for i_t in range(NT):
        # 存储当前隐藏状态
        p_h1 = tl.make_block_ptr(
            h + i_t * stride_h, (V, K), (K, 1), (i_v * BV, k_start), (BV, 64), (1, 0)
        )
        tl.store(p_h1, b_h1.to(p_h1.dtype.element_ty), boundary_check=(0, 1))
        # 加载权重并计算新值
        b_w = w_desc.load([i_t * BT, k_start])
        b_v = tl.dot(b_w, tl.trans(b_h1).to(b_w.dtype))
        b_v = v_desc.load([i_t * BT, i_v * BV]) - b_v
        # ... 门控（g, gk）和状态更新省略
```

python/sglang/srt/hardware_backend/xpu/kernels/fla/chunk_fwd.py

新增 XPU 专用的 chunk_gated_delta_rule_fwd_intra kernel，采用低寄存器 KKT 解法，分两遍计算对角和非对角块。

```
# 低寄存器版本：一次只保持一个 [BC, BC] 累加器，最小化寄存器压力
# Pass 1：对角块，每个子块一个 K 循环
for i_b in tl.static_range(4):
    i_tci = i_tc0 + i_b * BC
    b_A = tl.zeros([BC, BC], dtype=tl.float32)
    for i_k in range(tl.cdiv(K, BK)):
        p_k = tl.make_block_ptr(
            k, (T, K), (Hg * K, 1), (i_tci, i_k * BK), (BC, BK), (1, 0)
        )
```

```
            b_k = tl.load(p_k, boundary_check=(0, 1))
            b_A += tl.dot(b_k, tl.trans(b_k))
        # 门控、下三角约束和前向替换
        # ...
        # 存储对角结果
    # Pass 2：非对角块，按距离 d=1..3 逐对处理
    for d in tl.static_range(1, 4):
        for j in tl.static_range(0, 4 - d):
            i = j + d
            # 加载 Ai_ii、A_ij_raw、Ai_jj
            # 计算校正项并存储到 A 的上三角暂存区
```

评论区精华

Review 中主要讨论了以下几点：

- 目录结构：mingfeima 建议参照 #23654 将 vendor 特定 kernel 统一放入 hardware_backend/xpu/，作者采纳。
- BV 硬编码：mingfeima 指出 BV=16/32 应抽象为工具函数，经讨论后增加 get_block_size_v_cap()。
- block_ptr workaround：最初通过条件分支避免 block_ptr，后因 triton-xpu 修复而简化，但仍保留 tensor descriptor 路径作为性能优化。
- 设备检测：建议使用 get_device() 统一设备获取，已应用于测试文件。
- 性能分析：讨论使用 map 简化 profile 排序字段选择，但作者指出 profile_activities 是列表，最终保持 if-elif-else。
- BV 硬编码与抽象 (design): 将 BV 上限抽象到 utils 函数 get_block_size_v_cap()，避免 if-else 散落。
- block_ptr workaround (design): 不再需要 workaround，但保留 tensor descriptor 路径。
- 测试设备检测 (style): 作者采纳，更新测试文件。
- 目录结构对齐 (design): 按建议重构，XPU kernel 统一置于 hardware_backend/xpu/kernels/fla/。

风险与影响

- 风险：
  - triton-xpu 版本依赖：XPU 功能依赖特定版本的 triton-xpu（通过额外 index 安装），若未来更新可能引入兼容性问题。
  - kernel 数值一致性：新增的 XPU FLA kernel 与 CUDA kernel 可能产生微小数值差异，虽已通过 GSM8K 验证，但更细粒度的逐层比较未覆盖。
  - 维护成本：is_intel 条件分支分布在多个文件（chunk.py, kda.py, gdn_triton.py 等），未来需与 CUDA 路径保持同步，增加维护复杂度。
  - 性能风险：XPU kernel 参数（如 BV=16）是针对 Intel GPU 调优的，若在其他非 CUDA 设备上意外启用可能产生次优性能。
- 影响：

- 用户：Intel GPU 用户终于可以运行 Qwen3.5 模型，但需使用文档中指定的 triton-xpu 版本。
- 系统：新增 hardware_backend/xpu/ 模块，包含 ~700 行 XPU 专用 kernel；公共注意力路径增加了条件分发逻辑。
- 团队：Intel GPU 相关的 kernel 维护工作独立化，但需要与 FLA 社区保持同步。
- 风险标记：triton-xpu 版本依赖，is_intel 分支维护，kernel 数值差异风险

关联脉络

- PR #23654 Migrate flashinfer_cutedsl + DeepEP to MoeRunner: reviewer mingfeima 建议参照此 PR 的目录结构模式，将 vendor 特定 kernel 放入 hardware_backend。