PR #35337 完整报告

sgl-project/sglang

[XPU][CI] key persistent JIT kernel cache by image content ID

合并时间：2026-08-20 10:28

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35337

执行摘要

- 一句话：XPU CI 缓存按镜像 ID 键控，修复 dlopen 过期 .so 失败
- 推荐动作：值得精读，尤其适合 XPU/Intel CI 维护者。重点看 xpu_ci_start_container.sh 对 set -euo pipefail、root-owned 文件与容器化清理的组合处理，以及缓存键控的 fallback 设计；GDN kernel 的 valid_state 守卫可对照 PR #33431 的测试理解，是一个典型的「测试驱动 kernel 边界修复」案例。

功能与动机

PR body 明确指出：Nightly image rebuilds swap the runtime libs that cached JIT .so files link against（例如 libsycl.so.8 -> libsycl.so.9），Triton 只按源码 hash 键控缓存，旧 gpu-mask 缓存路径会 dlopen 旧镜像的过期 .so，导致每个 XPU CI job 以 OSError: libsycl.so.N: cannot open shared object file 死亡。此外，commit 6340656 说明 PR #33431 新增的 test_padded_state_index_is_skipped[all_padded] 在 XPU 上确定性失败：所有行均为 -1 哨兵时，哨兵进入指针算术并访问 state pool 之前的内存，pool 被改写。

实现拆解

1. 缓存路径重新键控（scripts/ci/xpu/xpu_ci_start_container.sh）：XPU_KERNEL_CACHE_DIR 显式覆盖时原样使用、不做剪枝；否则通过 docker image inspect --format '{{.Id}}' 取镜像 ID 的 12 位短 hash，拼出 kernel-cache-gpu${ZE_AFFINITY_MASK}-${IMG_ID_SHORT} 目录；inspect 失败时 fallback 到 kernel-cache-gpu${ZE_AFFINITY_MASK}-unversioned-$$ 一次性目录并打印警告。这样镜像内容变化即换新缓存，避免 stale .so 被 dlopen。
2. 旧缓存清理与超限重置统一改用 busybox（同一脚本）：用 shopt -s nullglob 收集同 GPU 下其他镜像 ID 的兄弟缓存目录及 legacy 无版本目录，通过 docker run --rm -v ${CACHE_ROOT}:/c busybox:latest rm -rf ... 清理 root 所有者文件（与 workspace 所有权重置同模式），|| true 兜底；容量超过 XPU_KERNEL_CACHE_MAX_MB（默认 5120 MiB）时，也将原 runner 级 rm -rf 改为同一个 busybox helper 清理子目录。
3. 放宽 stage-b 测试窗口（.github/workflows/pr-test-xpu.yml）：Run stage-b tests 步骤 timeout-minutes 从 60 提升到 120，避免 action 墙切断 test_intel_xpu_backend.py 的重试和 test_xpu_graph.py 的长时间执行。
4. 修复 GDN kernel 哨兵越界（python/sglang/srt/hardware_backend/xpu/kernels/fla/chunk_delta_h.py）：在 chunk_gated_delta_rule_fwd_kernel_h_blockdim64_k_loop 中读取

initial_state_indices 后计算 valid_state = index >= 0，Phase 1/Phase 2 的 h 加载分支以及 INPLACE_UPDATE 写回均以 valid_state 守卫，padded 行直接使用零块，不再让 -1 哨兵进入指针算术。

5. 测试配套：本 PR 未新增测试文件；kernel 修复的验证依赖 PR #33431 引入的 test_padded_state_index_is_skipped（该测试在 XPU 上原本确定性失败），CI 整体效果通过 XPU stage-b 全量回归验证。

关键文件：

- scripts/ci/xpu/xpu_ci_start_container.sh（模块 CI 缓存；类别 infra；类型 infrastructure）：核心变更文件：缓存键控从 gpu-mask 改为 gpu-mask + 镜像内容 ID，引入 busybox 清理 root 所有的旧缓存，并保留显式覆盖、fallback 与容量超限重置逻辑。
- python/sglang/srt/hardware_backend/xpu/kernels/fla/chunk_delta_h.py（模块 XPU 内核；类别 source；类型 core-logic；符号 chunk_gated_delta_rule_fwd_kernel_h_blockdim64_k_loop）：运行时 kernel 修复：为 GDN chunked kernel 增加 valid_state 守卫，避免 -1 padding 哨兵进入指针算术导致 XPU 上读写 state pool 之前的内存；是 PR #33431 测试在 XPU 上确定性失败的根因修复。
- .github/workflows/pr-test-xpu.yml（模块 工作流；类别 infra；类型 infrastructure）：将 stage-b 测试 action 超时从 60 分钟提高到 120 分钟，避免测试重试或挂起时被 wall 切掉（曾切断 test_intel_xpu_backend.py 重试与 test_xpu_graph.py 的长时间执行）。

关键符号：chunk_gated_delta_rule_fwd_kernel_h_blockdim64_k_loop

关键源码片段

scripts/ci/xpu/xpu_ci_start_container.sh

核心变更文件：缓存键控从 gpu-mask 改为 gpu-mask + 镜像内容 ID，引入 busybox 清理 root 所有的旧缓存，并保留显式覆盖、fallback 与容量超限重置逻辑。

```
# 显式覆盖优先：使用 XPU_KERNEL_CACHE_DIR 原样路径，不做剪枝
if [[ -n "${XPU_KERNEL_CACHE_DIR:-}" ]]; then
  XPU_KERNEL_CACHE_HOST="${XPU_KERNEL_CACHE_DIR}"
else
  # `|| IMG_ID_SHORT=""` 保证 pipefail 下 inspect 失败不会杀死脚本
  IMG_ID_SHORT=$(docker image inspect --format '{{.Id}}' "${IMAGE}" 2>/dev/null \
    | sed 's/^sha256://' | cut -c1-12) || IMG_ID_SHORT=""
  CACHE_ROOT="${HOME}/.cache/sglang-xpu-ci"
  GPU_KEY="gpu${ZE_AFFINITY_MASK:-shared}"
  if [[ -n "${IMG_ID_SHORT}" ]]; then
    # 缓存目录包含镜像内容 ID：镜像重建后自动换新缓存，避免 dlopen 旧 .so
    XPU_KERNEL_CACHE_HOST="${CACHE_ROOT}/kernel-cache-${GPU_KEY}-${IMG_ID_SHORT}"
    # 清理同 GPU 下其他镜像 ID 的兄弟缓存及旧版无版本目录（文件为容器 root 所有）
    shopt -s nullglob
    stale_siblings=("${CACHE_ROOT}"/kernel-cache-"${GPU_KEY}"-* "${CACHE_ROOT}/kernel-
    cache-${GPU_KEY}")
    shopt -u nullglob
    for sibling in "${stale_siblings[@]}"; do
      [[ -d "${sibling}" ]] || continue
```

```
      [[ "${sibling}" == "${XPU_KERNEL_CACHE_HOST}" ]] && continue
      echo "Pruning stale kernel cache: ${sibling}"
      docker run --rm -v "${CACHE_ROOT}:/c" busybox:latest \
        rm -rf "/c/$(basename "${sibling}")" || true
    done
  else
    # inspect 失败时使用一次性目录，避免读到被污染的旧缓存
    echo "Warning: could not resolve image ID for ${IMAGE}; using throwaway cache." >&2
    XPU_KERNEL_CACHE_HOST="${CACHE_ROOT}/kernel-cache-${GPU_KEY}-unversioned-$$"
  fi
fi
```

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

运行时 kernel 修复：为 GDN chunked kernel 增加 valid_state 守卫，避免 -1 padding 哨兵进入指针算术导致 XPU 上读写 state pool 之前的内存；是 PR #33431 测试在 XPU 上确定性失败的根因修复。

```
# Padded 行携带 -1 哨兵值；若不守卫，哨兵会进入指针运算，
# 让地址落到 state pool 之前，导致越界读写在 XPU 上改变状态池。
valid_state = index >= 0
h0 = initial_state + index * stride_h
ht = initial_state + index * stride_h
if USE_INITIAL_STATE:
    h0 = h0 + i_h * V * K
if INPLACE_UPDATE:
    ht = ht + i_h * V * K

# 主递推循环：时间作为外层循环
for i_t in range(NT):
    # Phase 1: 计算 v_new = u - sum_k(w_k @ h_k^T)
    b_v_corr = tl.zeros([BT, BV], dtype=tl.float32)
    for k_blk in range(0, K, 64):
        if i_t == 0:
            if USE_INITIAL_STATE and valid_state:
                p_hs = tl.make_block_ptr(
                    h0, (V, K), (K, 1), (i_v * BV, k_blk), (BV, 64), (1, 0)
                )
                b_h = tl.load(p_hs, boundary_check=(0, 1)).to(tl.float32)
            else:
                # 无效行（padded）直接置零，不触碰初始状态池
                b_h = tl.zeros([BV, 64], dtype=tl.float32)
        elif valid_state:
            p_hs = tl.make_block_ptr(
                ht, (V, K), (K, 1), (i_v * BV, k_blk), (BV, 64), (1, 0)
            )
            b_h = tl.load(p_hs, boundary_check=(0, 1)).to(tl.float32)
        else:
            b_h = tl.zeros([BV, 64], dtype=tl.float32)
```

```
        # 存储更新前的 h 到输出
        p_ho = tl.make_block_ptr(
            h + i_t * stride_h,
            (V, K),
            (K, 1),
            (i_v * BV, k_blk),
            (BV, 64),
            (1, 0),
        )
        tl.store(p_ho, b_h.to(p_ho.dtype.element_ty), boundary_check=(0, 1))

        # 累加 w_k @ h_k^T 修正项
        b_w = w_desc.load([i_t * BT, k_blk])
        b_v_corr += tl.dot(b_w, tl.trans(b_h).to(b_w.dtype))
    # 后续 gating 与 Phase 2 更新逻辑与 Phase 1 对称，同样受 valid_state 守卫
```

评论区精华

本 PR 无 review 评论（comments 与 review comments 均为 0），两位 reviewer pramodkumar-habanalabs 与 mingfeima 直接 approve。有价值的决策讨论体现在 6 个 commit 的演进中：

- commit 5011ff6 指出：在 set -euo pipefail 下，docker image inspect 失败会以 rc=1 穿过管道直接杀死脚本，导致 fallback 分支成为死代码、step 静默退出 1；修复方式是在管道后追加 || IMG_ID_SHORT="" 兜底。
- commit 73b3009 记录：60 分钟 action 墙切断了 test_intel_xpu_backend.py 的完整重试（1800s 首测 + 91s 重试），且 test_xpu_graph.py 在测试 10/11 开始挂起，因此将超时提到 120 分钟。
- commit 6340656 记录：test_padded_state_index_is_skipped[all_padded] 在 XPU 上让 GDN pool 发生变更，必须加 -1 哨兵守卫。
- image-id fallback 在 pipefail 下不可达 (correctness): 在管道后追加 || IMG_ID_SHORT="" 兜底，保证 inspect 失败时 fallback 分支可达。
- stage-b 60 分钟 wall 切掉测试 (performance): 将 Run stage-b tests 的 timeout-minutes 从 60 提升到 120。

风险与影响

- 风险：
  1. XPU kernel 修复缺独立测试文件：chunk_delta_h.py 的控制流改动无新增测试，依赖 PR #33431 的 test_padded_state_index_is_skipped 回归；若该测试未覆盖所有分支组合（如 USE_INITIAL_STATE 与 valid_state 的交叉），仍存在回归盲区。
  2. 缓存清理依赖 busybox 镜像：docker run busybox:latest 需要 runner 能拉取镜像且 docker 可用；失败时虽有 || true 兜底，但旧缓存不会被清理，长期可能堆积磁盘。
  3. 镜像重建后全量冷编译：镜像 ID 变化意味着所有 XPU 机器首次运行都要全量 JIT 编译，stage-b 超时虽已放宽到 120 分钟，但极端情况下仍有超时风险；docker image

inspect 失败时使用一次性目录，若持续失败则每次都是冷缓存。

4. shell 管道兼容性：set -euo pipefail 与 docker image inspect | sed | cut 管道交互曾导致 fallback 死代码 bug，虽然已修复，但后续任何管道改动都需注意 pipefail 语义。
5. 12 位镜像 ID 截断碰撞：sha256 截断 12 位理论上存在极低碰撞概率，作为 CI 缓存键可接受，但需知晓该假设。

- 影响：

1. XPU CI 稳定性：修复镜像重建后所有 XPU job 因 dlopen 旧 .so 必挂的问题；镜像不变期间缓存复用，镜像变更时自动换新并清理旧缓存，磁盘占用受 5 GiB cap 约束。
2. 运行时用户：非 XPU 后端不受影响；XPU 上 GDN（gated delta rule）chunked 前向在 padded 批次下的行为被纠正，避免状态池被越界读写导致的结果错误或潜在崩溃。
3. 团队维护成本：XPU CI 缓存语义更复杂（镜像 ID 维度），维护者需理解 XPU_KERNEL_CACHE_DIR 覆盖、pruning 与 fallback 的优先级；stage-b 窗口翻倍意味着 CI 资源占用上限提高。 - 风险标记：XPU 后端 kernel 守卫修复无独立测试文件，缓存清理依赖 busybox 镜像可拉取，镜像重建后全量冷编译，超时风险仍存，shell 管道 pipefail 兼容性（曾有死代码 bug）

关联脉络

- PR #33431 PR #33431 (commit 提及，引入 test_padded_state_index_is_skipped): commit 6340656 明确提到该 PR 新增的 test_padded_state_index_is_skipped[all_padded] 在 XPU 上确定性失败，驱动了 chunk_delta_h.py 的 -1 哨兵守卫修复。