PR #32832 完整报告

sgl-project/sglang

[AMD] [sgl-kernel] Bypass caches for peer traffic in ROCm custom all-reduce

合并时间：2026-08-21 06:24

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/32832

执行摘要

PR #32832 针对 ROCm 自定义 all-reduce（CAR v1）的 peer 通信流量做缓存绕过：peer 数据经 xGMI 读取后只消费一次，落入 L2 只会污染缓存。作者参考 RCCL op128.h，新增 load_bypass/store_bypass，通过 b128 builtins（系统 syncscope 置 sc0/sc1）或 nontemporal 64 位回退，使读写直达 HBM。改动仅影响 USE_ROCM 路径，CUDA 构建不变；在 gfx950 上 TP=2/4/8 保持位级确定性，<=8 MB 消息延迟降低 14%-33%。两个 allreduce 头文件均需同步修改，review 中已澄清原因并获批。

功能与动机

PR body 明确：custom all-reduce 每次经 xGMI 读取所有 peer 的 buffer，结果也只被消费一次，缓存这些单次数据只会污染 L2。因此要为 packed 128 位读写提供 cache-bypassing 路径。hubertlu-tw 在评论中补充：CAR v1 在 deterministic inference、SGLANG_USE_1STAGE_ALLREDUCE=1、SGLANG_USE_AITER_AR=false 且无 AITER 时启用，该优化能直接帮助 Miles 平台的 AMD GPU 用户。

实现拆解

变更入口：python/sglang/kernels/aot/csrc/allreduce/custom_all_reduce.cuh 与 custom_all_reduce_hip.cuh，纯新增代码（各 +103）。

1. 编译器能力检测：USE_ROCM 内定义 ROCM_HAVE_GLOBAL_DWORDX4_BUILTINS，通过 __has_builtin 同时检查 __builtin_amdgcn_global_load_b128 与 __builtin_amdgcn_global_store_b128；同时定义 sgl_v4u（unsigned int x4 扩展向量）与 sgl_v4u_gptr（address_space(1) 指针）。
2. 缓存绕过访问器：load_bypass/store_bypass 均为模板，要求 sizeof(P)==16。主路径：b128 builtins + 空 syncscope 字符串（系统内存域），编译为 global_{load,store}_dwordx4 并置 sc0/sc1 位。回退路径：两条 64 位 __builtin_nontemporal_{load,store}，置 SLC 位，与 RCCL op128.h 的非 builtin 路径一致。
3. 数据通路改造：packed_reduce 中读取 peer 元素改为 load_bypass，结果写入改为 store_bypass，使远程 xGMI 流量绕过 L2 直达 HBM。USE_ROCM 保护下 CUDA 路径原样保留。

4. 验证配套：未新增自动化测试；作者在 gfx950（MI35x）上通过 ISA 反汇编确认 sc0/sc1，并给出 TP=8、hidden=16384、bf16、10 trials 的基准；hubertlu-tw 本地复测 test_amd_deterministic_custom_allreduce.py 与 microbenchmark 后 approve。

关键源码片段

python/sglang/kernels/aot/csrc/allreduce/custom_all_reduce.cuh

CAR 主实现文件，新增缓存绕过访问器并改造 packed_reduce 的 peer 读写路径。

```
// custom_all_reduce.cuh / custom_all_reduce_hip.cuh 中新增的缓存绕过辅助函数
// 整个新增块由 USE_ROCM 保护，CUDA 构建不受影响。

template <typename P>
DINLINE P load_bypass(const P* ptr) {
    // 只支持 16 字节 packed 类型，例如 float4、int4 等
    static_assert(sizeof(P) == 16, "load_bypass expects a 16-byte packed type");
#if ROCM_HAVE_GLOBAL_DWORDX4_BUILTINS
    // 主路径：b128 builtins + 空 syncscope（系统内存域），置 sc0/sc1 绕过位
    union {
        P p;
        sgl_v4u v; // unsigned int x4 扩展向量类型
    } u;
    u.v = __builtin_amdgcn_global_load_b128((sgl_v4u_gptr)ptr, SGL_SYSTEM_SYNCSCOPE);
    return u.p;
#else
    // 回退路径：两条 64 位 nontemporal 访问，置 SLC 位，与 RCCL op128.h 一致
    union {
        P p;
        uint64_t u64[2];
    } u;
    const uint64_t* addr = reinterpret_cast<const uint64_t*>(ptr);
    u.u64[0] = __builtin_nontemporal_load(addr);
    u.u64[1] = __builtin_nontemporal_load(addr + 1);
    return u.p;
#endif
}

template <typename P>
DINLINE void store_bypass(P* ptr, const P& val) {
    static_assert(sizeof(P) == 16, "store_bypass expects a 16-byte packed type");
#if ROCM_HAVE_GLOBAL_DWORDX4_BUILTINS
    union {
        P p;
        sgl_v4u v;
    } u;
    u.p = val;
    __builtin_amdgcn_global_store_b128((sgl_v4u_gptr)ptr, u.v, SGL_SYSTEM_SYNCSCOPE);
#else
    union {
```

```
        P p;
        uint64_t u64[2];
    } u;
    u.p = val;
    uint64_t* addr = reinterpret_cast<uint64_t*>(ptr);
    __builtin_nontemporal_store(u.u64[0], addr);
    __builtin_nontemporal_store(u.u64[1], addr + 1);
#endif
}
```

评论区精华

- HaiShaw：Update to custom_all_reduce_hip.cuh seems not needed.
- hubertlu-tw：我们确实需要 hip.cuh 的改动，因为 CAR v1 不依赖 hipification；并列出 CAR v1 的启用条件，指出该 PR 可帮助 Miles 用户。
- HaiShaw：HIP specific changes（确认这是 HIP 专属改动）。
- hubertlu-tw：已在本地用 test_amd_deterministic_custom_allreduce.py 和 microbenchmark 验证，LGTM。
- HaiShaw 还请求提供运行 CAR v1 的完整命令示例（评论被截断），该请求在评论区未见完整回应。

风险与影响

- 影响面窄且可控：改动全部在 USE_ROCM 保护下，CUDA 构建与现有行为不变。
- 双文件维护成本：cuh 与 hip.cuh 同步新增同一段逻辑，未来改动若只改其一会导致失效。
- 工具链依赖：主路径依赖 AMD clang 的 b128 builtins；旧工具链走 nontemporal 回退，绕过效果与带宽表现可能不如主路径。
- 验证局限：仅在 gfx950 单机验证，未覆盖 MI200、跨节点或更大 TP；且 PR 的 CI 状态（pr-test、pr-test-extra）显示失败，合并前需确认失败原因是否相关。
- 无新增自动化测试是主要短板，后续回归依赖手工脚本。

关联脉络

该 PR 是 ROCm all-reduce 性能优化的一环，与确定性推理特性线相交：CAR v1 仅在 deterministic 推理时启用，同仓库 PR #35632（GDN prefill 保持 Triton 确定性）同属该路径的稳定性维护。实现层面直接借鉴 RCCL op128.h 的缓存绕过模型，属于 SGLang 将 RCCL 经验下沉到自有 kernel 的典型手法；当前仓库历史 PR 中没有直接修改这两个文件的先例，因此这是一次独立的性能专项，后续可考虑为 CAR 增加自动化基准测试。