https://crad.ict.ac.cn
62(6): 1333−1346, 2025

SW-IntraCC: 一种面向申威智能加速卡内部的集合通信机制

赵玉龙[1] 顾燕卿[1] 田松涛[1] 吴春志[2] 汤凌韬[1] 张鲁飞[1] 秦晓军[1] 刘 鑫[3] 陈左宁[3]

[1](数学工程与先进计算国家重点实验室 江苏无锡 214125)

[2](航天工程大学 北京 100004)

[3](国家并行计算机工程技术研究中心 北京 100083)

(yyylx@263.net)

SW-IntraCC: A Collective Communication Mechanism for the Internals of Sunway AI Acceleration

Zhao Yulong[1], Gu Yanqing[1], Tian Songtao[1], Wu Chunzhi[2], Tang Lingtao[1], Zhang Lufei[1], Qin Xiaojun[1], Liu Xin[3], and Chen Zuoning[3]

[1](*State Key Laboratory of Mathematical Engineering and Advanced Computing*, *Wuxi*, *Jiangsu* 214125)

[2](*Space Engineering University*, *Beijing* 100004)

[3](*National Parallel Computer Engineering Technology Research Center*, *Beijing* 100083)

Abstract The number of large-scale language model parameters is growing exponentially, which puts forward higher requirements on the arithmetic density and communication efficiency of the acceleration card, and promotes the rapid development of new architectures, such as single-card multi-core, multi-chip and multi-communication entities. Sunway AI acceleration card adopts four-core group on-chip Ring bus architecture, but in the large model training, the data communication volume is large, and the traditional Ring collection communication method faces the core bottlenecks such as the double limitation of single-core group memory capacity and transmission bandwidth, low collection communication efficiency, and the inability to overlap the communication and computation. In this paper, the optimization framework of SW-IntraCC (Sunway-intra collective communication) is proposed by adopting the concept of software-hardware collaborative design to break through the above limitations through the three-tier storage architecture. First, the three-tier storage architecture is constructed based on on-chip high-speed Ring network, which expands the memory capacity of a single core group by up to four times and increases the host-accelerator card transmission bandwidth by 2.5 times; Second, an intra-chip cross shared communication (CSC) algorithm is designed with interleaved memory access patterns, implementing CSC-AG (CSC-AllGather) and CSC-RS (CSC-ReduceScatter) operators optimized for large model training. Benchmark results demonstrate that CSC achieves 2.15 times higher communication efficiency compared with conventional collective primitives. Finally, a bidirectional operator fusion strategy is proposed to enable communication-computation overlap, yielding a 59% improvement in communication performance after optimization.

Key words collective communications; Sunway AI acceleration card; SW-IntraCC; communications optimization; ring network

摘 要 大规模语言模型参数量呈指数级增长趋势,对加速卡算力密度与通信效率提出更高要求,推动单卡多芯粒、多芯片及多通信实体等新型架构的快速发展.申威智能加速卡采用4个核组片上环网架构,但在大模型训练中,数据通信量大和卡内传统Ring集合通信方式面临单核组显存容量与传输带宽双重限制、

收稿日期: 2025-03-01; **修回日期**: 2025-04-08

卡内集合通信效率低、通信计算无法重叠等核心瓶颈. 采用软硬协同设计理念提出 SW-IntraCC (Sunway-intra collective communication) 的优化框架, 通过三级存储架构突破上述限制. 首先, 基于片上高速环网构建三级存储架构, 单核组显存容量最高扩大至 4 倍, 主机-加速卡传输带宽提升 2.5 倍; 其次, 设计采用交叉共享访存的片内高效 CSC (cross shared communication) 通信算法, 实现面向大模型训练的典型通信算子 CSC-AG (CSC-AllGather) 和 CSC-RS (CSC-ReduceScatter), 通信效率是传统方式的 2.15 倍; 最后, 提出双向算子融合的通信计算重叠方法, 实现通信与计算重叠, 优化后通信性能提升 59%.

关键词 集合通信; 申威智能加速卡; SW-IntraCC; 通信优化; 环形网络

中图法分类号 TP311.13; TP309

DOI: 10.7544/issn1000-1239.202550143 **CSTR:** 32373.14.issn1000-1239.202550143

人工智能技术的快速发展推动大规模预训练模型参数量突破千亿级, 其训练推理过程对计算资源、存储带宽及通信系统构成三重挑战[1]. 以 DeepSeek 为代表的大模型[2]在分布式训练中面临显存容量、实时性及多卡协同效率等关键约束, 促使分布式多机多卡集群训练架构[3]成为主流解决方案. 集合通信技术[4]虽有效支撑跨卡数据交互, 但在大规模扩展时面临传输效率衰减与通信开销非线性增长等瓶颈问题[5], 催生出高密度集成加速卡等技术路线, 单卡多芯粒、多芯片等集成技术成为研究热点.

然而, 此类架构在提升并行度的同时, 导致通信拓扑复杂度显著增加: 1)传统 Ring 通信依赖 PCIe 总线绕经主机中转, 引入额外延迟[6]; 2)片内多实体通信受限于 PCIe 带宽及通道竞争, 通信效率低下[7-9]; 3)显存碎片化问题使单个通信实体可用容量受限, 制约十亿级参数模型部署[10]. 当前研究主要聚焦跨节点通信协议优化[11-12], 而节点内多通信实体间(Die 内/芯片间/卡内多个逻辑或物理通信单元)的高效通信机制仍缺乏系统性探索, 成为制约新型加速卡效能释放的关键瓶颈.

申威智能(SW-AI)加速卡采用单卡四核组环网架构, 内部直接内存访问带宽高、存储层次丰富、端到端延迟低, 为卡内通信优化提供硬件基础. 然而, 在 LLaMa 70B 模型训练中, Ring 集合通信[3]将 4 个核组作为 4 个通信实体, 暴露以下问题: 1)卡内 4 个核组间的 AllGather 和 ReduceScatter 操作实际带宽为环网带宽的 30% 左右, 均远低于环网的设计带宽; 2)计算与通信串行化导致计算资源闲置率不小于 50%; 3)单核组显存上限 15 GB, 需频繁触发主机-加速卡数据传输. 上述问题使 128 卡集群训练中通信开销占比超 20%, 严重限制系统有效算力. 针对上述挑战, 本文提出基于申威智能加速卡软件生态的 SW-IntraCC 通信优化框架, 其主要贡献有 3 点:

1)基于片上环网构建 L0(局部高速存储)-L1(Local-HBM)-L2(Global-HBM)层次化三级存储架构, 在 SDAA[13]运行时系统中实现了三级存储共享显存管理机制; 单核组可访问的最大显存容量是原来的 4 倍, 从 15 GB 提升至 60 GB; 主机-加速卡之间的数据传输带宽实现 3 倍的提升.

2)基于三级存储架构, 本文提出采用交叉共享访存的片内通信算法, 设计实现 CSC-AG、CSC-RS 等集合通信算子, 该算法充分利用了环网总线直接交叉段内存访问的特性, 减少通信开销, 提高数据传输的并行性, 性能是 Ring 集合通信方式的 2.15 倍.

3)针对通信计算无法重叠问题, 将通信算子与前后计算算子深度融合, 利用局部高速存储空间绕过主存中转环节, 实现双向算子融合的通信计算重叠方法, 完成了计算与通信之间的高效重叠, 提高了实体间的通信效率; 通过通信与计算的重叠, 性能较重叠前提升了 59%.

在接下来的章节中, 将详细介绍本文的研究背景与研究动机, SW-IntraCC 机制的设计原理、实现方法以及性能评估结果, 以展示其在提升申威智能加速卡内部通信效率方面的有效性和优势.

1 卡内通信研究背景与动机

1.1 多通信实体现状

现代芯片设计已进入“后摩尔定律”时代, 设计技术正朝着多芯粒[11]方向发展. 硬件厂商通过将多个独立的通用计算 Die、AI 加速芯粒等集成到一块芯片中, 并借助高速互连技术实现协同工作, 单芯片的计算能力得以显著提升. 在此基础上, 多个相同功能的芯片还可集成于单张加速卡, 从而进一步提高单卡的计算能力.

基于现代芯片设计发展趋势, 华为、寒武纪等公

司相继推出集成多个人工智能芯片的加速卡. 此类加速卡采用独特的设计, 对用户呈现出多个独立的加速设备(通信实体). 如图 1 所示, 华为 Atlas 300I[12] 推理卡配备了 4 个 Ascend 310 芯片, 芯片之间通过 PCIe 总线实现数据通信和任务协作. 为了实现 4 个加速实体间的高效通信, 华为提供了基于 Ring 和 mesh+Ring 等算法的集合通信库 HCCL(Huawei collective communication library). 寒武纪 MLU370-X8[13] 加速卡采用了双芯片多 Die 设计方法, 芯片间通过 MLU-Link, PCIe 等多种高速总线互联, 为支持卡内 2 个加速实体之间的高效通信, 寒武纪提供了专用的集合通信库 CNCL(Cambricon communications library), 该库支持基于 PCIe, MLU-Link, RoCE[14] 和 IB 等多种互联技术的通信传输.

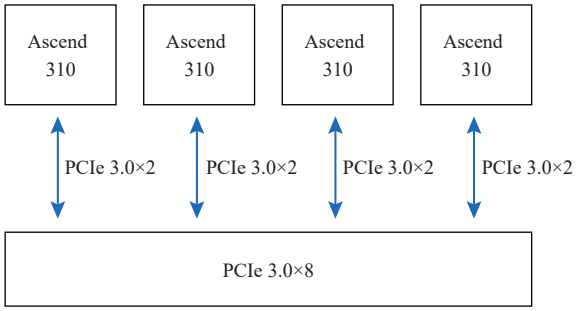

Fig. 1 Structure of Huawei Ascend 310 accelerator card

图 1 华为昇腾 310 加速卡结构

申威智能加速卡通过 PCIE-X16 与主机连接, 内置新一代申威异构 AI 处理器[13], 该处理器采用片上多核组、通专结合计算核心、脉动阵列、乘加运算单元相结合的多级瓦片架构, 并基于申威研发的自主 SW64 指令集系统, 支持 32 b 定长指令和 64 b 字长, 能满足多种计算需求. 处理器逻辑结构如图 2 所示, 通过运行时系统向用户空间抽象出 4 个逻辑独立的加速设备(通信实体), 采用多通信实体架构设计, 集成 1 个管理控制核心 MPE(management processing element), 又称主核; 4 个从核阵列(Core Group), 又称核组、通信实体; 以及片上环网总线(Ring Bus). 其中, MPE 作为主控单元负责系统级调度与 I/O 资源管理; 从核阵列作为计算单元, 每个阵列包含 32 个运算核心 CPE (computing process element), 又称从核, 以 4×8 二维阵列拓扑结构实现并行加速, 并配备局部数据存储器(local data memory, LDM)与高带宽显存(high bandwidth memory, HBM); 环网总线实现 MPE 和核组间的高速互连, 支持全局交叉段共享, 又称交叉段、大共享段机制. 存储系统采用多级层次化架构: LDM 与 CPE 紧耦合, 容量为 KB 级, 带宽 TB 级; 本地 HBM 为阵列内部共享, 容量 15 GB 左右; 全局交叉段 HBM 通过环网实现跨核组共享, 容量可扩展至 60 GB. CPE 可以通过 ld/st 访存指令直接访问主存空间, 还可以通过 DMA(direct memory access)实现 LDM 和高速显存之间的批量数据传输, DMA 传输的效率与传输的数据量、DMA 命令数量、数据在主存中的连续性以及 DMA 传输方式等密切相关. 此外, 运算核心阵列还支持 LDM 共享和 RMA(remote memory access)两种机制以实现阵列上 LDM 间的数据交互. 该架构通过硬件级计算-通信解耦设计, 为大规模并行任务提供高吞吐、低延迟的数据通路支撑.

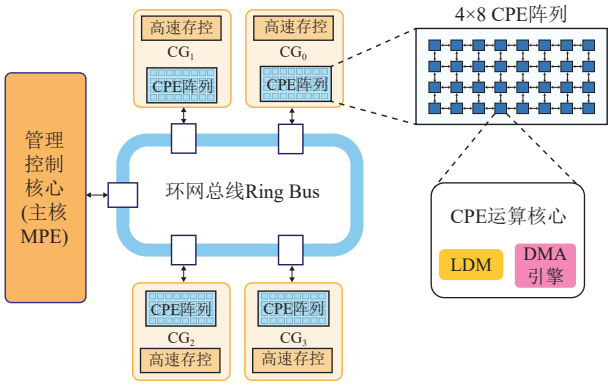

Fig. 2 Sunway AI processor architecture

图 2 申威 AI 处理器架构

1.2 大模型训练通信场景

在 Transformer[15] 架构的大模型训练过程中, 随着模型规模和训练数据量的急剧增长, 单机单卡训练模式已无法满足算力需求, 分布式训练成为主流解决方案. 当前主流的分布式并行策略包括数据并行(data parallelism, DP)[16]、张量并行(tensor parallelism, TP)[17]、流水并行(pipeline parallelism, PP)[18-19]、序列并行(sequence parallelism, SP)[20] 等.

以国产申威智能加速卡分布式训练场景为例, 在 LLaMa 70B 大模型训练场景中, 硬件配置采用 16 台服务器组成的集群, 每台服务器搭载 8 张申威智能加速卡, 共计 128 张加速卡. 为优化计算效率, 采用混合并行策略组合: 集群层面实施流水并行, 并行度 pp=16; 节点层面采用数据并行, 并行度 dp=8; 单卡层面 4 个高带宽通信实体组成张量并行组, 并行度 tp=4, 同时开启序列并行[20], 加速卡内 TP+SP 并行方案如图 3 所示.

为了保证单卡内核组的高效通信和同步, 通常采用 2 种集合通信操作 Ring-AG (Ring AllGather)和 Ring-RS(Ring ReduceScatter). AllGather[21] 是实现跨节

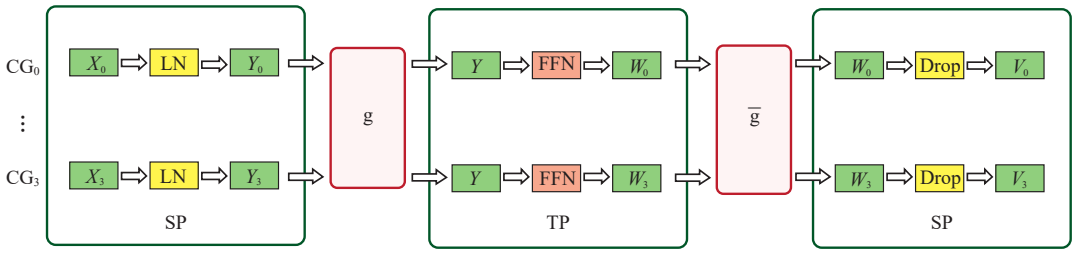

Fig. 3 TP+SP parallel scheme in the accelerating card

图 3 加速卡内 TP+SP 并行方案

点数据全局同步的集合通信操作. 各计算节点通过互连网络交换本地数据, 最终所有节点持有多节点数据的完整副本. 在分布式训练中, AllGather 通常用于汇聚各节点的梯度信息, 确保所有节点使用相同的模型参数[22]. ReduceScatter[23] 是一种将全体计算节点的数据分散并减少的操作. 每个节点计算部分结果并发送给其他节点, 最终节点会汇总并得到一个聚合值. 在深度学习中, ReduceScatter 主要用于将大规模数据集进行分散计算以减少单一节点的计算负担. 这 2 个操作为共轭操作, 如图 3 所示, 在前向传播中, g 对应 Ring-AG 操作, 而在反向传播中对应 Ring-RS 操作; 相反, ḡ在前向传播中对应 Ring-RS 操作, 而在反向传播中对应 Ring-AG 操作.

加速卡内部采用 TP+SP 并行模式后通信量的计算如式(1)(2)所示:

$$V_{\text{TP-AG}} = 6\times fp_{\text{SIZE}}\times\frac{bs}{dp}\times s\times h\times\frac{l}{pp} = 120\ \text{GB}, \quad (1)$$

$$V_{\text{TP-RS}} = 6\times fp_{\text{SIZE}}\times\frac{bs}{dp}\times s\times h\times\frac{l}{pp} = 120\ \text{GB}. \quad (2)$$

在式(1)中, $V_{\text{TP-AG}}$表示在一次全激活重计算训练中, 4 个通信实体进行 Ring-AG 的通信量. 其中, “6”表示每层正向和反向传播总计有 6 次通信; fp_{SIZE}为单个浮点数占用字节数(2 个字节); 总批大小 bs=2 048; dp 是数据并行粒度; 隐藏维度 h=8 192, 模型层数 l=80. 而式(2)中的 $V_{\text{TP-RS}}$表示 Ring-RS 的通信量.

上述计算结果表明, 加速卡内部通信实体需要频繁进行 Ring-AG 和 Ring-RS 操作, 该过程涉及大规模数据传输. 然而, 实际使用中, 数据量为 8 MB 时, Ring-AG 的算法带宽为 34.1 GB/s, Ring-RS 的平均数据量为 32 MB, 算法带宽为 54.9 GB/s, 均远低于环网设计带宽, 整体训练中通信开销占比超 20%, 加速卡内的集合通信耗时长, 已成为训练过程的瓶颈, 极大限制了基于申威智能加速卡的大规模模型训练性能.

1.3 性能瓶颈分析

本节以图 4 所示的基于 Ring 算法实现的 AllGather

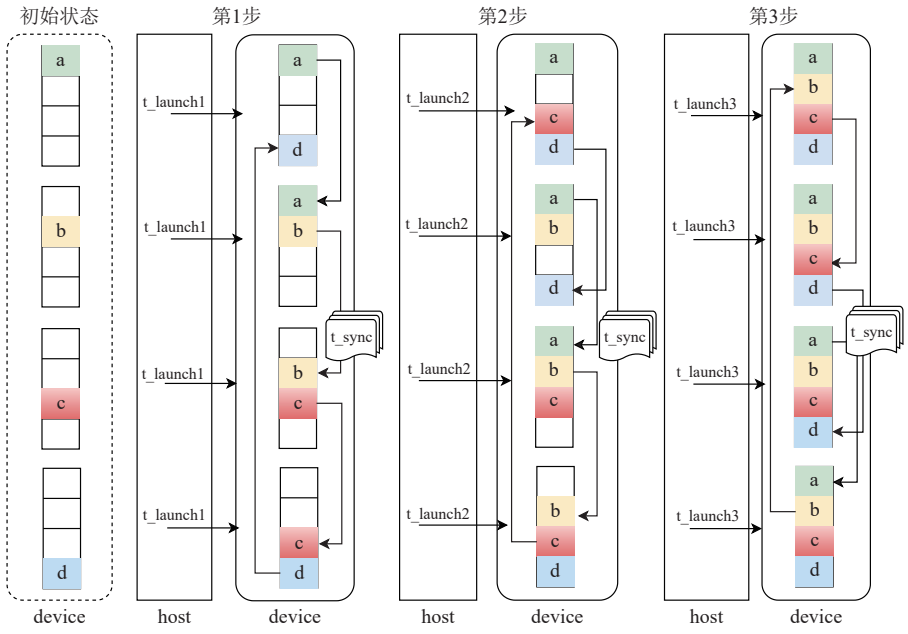

Fig. 4 Illustration of communication of Ring-AG algorithm

图 4 Ring-AG 算法的通信示意图

传输模型为例，系统分析申威智能加速卡内部集合通信面临单核组显存容量与传输带宽双重限制、卡内集合通信效率低及通信计算无法重叠等核心瓶颈三大问题.

首先，卡内集合通信效率低问题. 如图 4 所示，在 AllGather 操作中，主机侧首先发起通信请求 launch1，其时间开销 t_launch1 包含指令下发与参数配置延迟；随后各核组通过从核 DMA 执行数据拷贝操作；完成数据迁移后需通过 PCIe 总线进行全局同步，时间开销为 t_sync. 采用环网拓扑的 AllGather 集合通信需要迭代执行 3 次上述流程，最终实现全量数据同步，如图 4 中 a、b、c、d 数据聚合过程所示.

$$BW_{\mathrm{AG}}=\frac{V_{\mathrm{AG}}}{3\times\left(t_{\mathrm{LAUNCH}}+t_{\mathrm{SYNC}}+\dfrac{V_{\mathrm{AG}}}{4\times BW_{\mathrm{Ring}}}+t_{\mathrm{OTHERS}}\right)}=41\ \mathrm{GB/s}. \tag{3}$$

根据式(3)Ring-AG 算法的带宽上限模型，当单次 AllGather 传输量V_{AG}为 8 MB 和环网物理带宽BW_{Ring}为 100 GB/s 时，结合四核组并行启动特性，其理论最大算法带宽可估算为 41 GB/s. 该值显著低于理论物理带宽，主要源于启动延迟t_{LAUNCH}约为 25 μs，同步开销t_{SYNC}为 15 μs，其他时间开销t_{OTHERS}为 5 μs 的累计影响.

其次，通信计算无法重叠问题. 在分布式计算架构中，计算与通信的流水线重叠(overlap)[24]是性能优化的核心策略. 然而申威智能加速卡的从核 DMA 引擎需要同时服务于计算算子(将数据从 HBM 加载到 LDM)和集合通信算子(核组间数据传输)，导致硬件资源竞争. 这种资源共享机制使得计算与通信操作无法并行执行，通信时延无法被计算时间掩盖. 这种分离的计算与通信模式不仅增加了通信延迟，而且降低了整体运行效率.

最后，单核组显存容量与传输带宽双重限制问题.1)显存容量约束问题. 单核组可用显存容量上限为 15 GB. 虽能满足中小规模计算需求，但在大模型推理场景下，该容量限制迫使系统频繁执行多核组分布式通信，引发额外的内存交换开销，导致计算延迟增加和推理效率降低. 2)主机-加速卡带宽竞争问题. 4 个核组共享 PCIe 总线与主机通信，在传输模型参数等关键数据时，各核组实际可用带宽仅为总带宽的 1/4. 这种带宽分配机制在大规模并行计算场景下，尤其是高密度数据交换任务中，严重制约了加速卡的性能表现.

综上，申威智能加速卡的四核组架构虽提升了本地显存带宽利用率，但在处理大模型推理等高内存需求任务时，仍面临主机-设备传输带宽及显存容量约束、通信带宽受限、通信计算无法重叠三大关键挑战. 这些问题导致计算延迟增加和整体效率下降，限制了其在 AI 领域的应用潜力. 为此，本文提出基于三级存储架构的新型卡内集合通信机制，旨在优化申威智能加速卡内多通信实体间的数据传输效率.

2 SW-IntraCC 设计实现

针对申威智能加速卡内部 Ring 集合通信存在的通信效率低、计算与通信无法重叠执行、本地显存容量及主机-加速卡数据传输受限等问题. 本文运用软硬协同设计的理念，提出 SW-IntraCC 的设计方案. 该方案基于芯片环网特性设计出三级存储架构，提升单核组可用显存空间，有效缓解了主机-加速卡数据传输瓶颈；在三级存储架构上采用交叉共享访存的片内通信算法和双向算子融合的通信计算重叠方法，显著降低加速卡内通信开销，提升集合通信效率，并实现通信与计算高效重叠.

2.1 基于片上环网的三级存储架构

SW-AI 处理器采用异构多核组架构设计，其硬件包含 4 个核组及对应的 HBM 控制器. 传统二级存储架构(LDM+HBM)中，每个核组独立使用，从存储空间视图来看，包含 LDM 和 HBM 两个存储层次，如图 5 所示. 该架构在 4 个核组之间无交互时能够最大化利用本地 HBM 的性能，能满足早期的应用需求，但在面对大模型推理和训练任务时，单核组显存容量不足、通信效率低以及数据传输存在瓶颈等问题逐渐凸显，限制了申威智能加速卡在高性能计算和人工智能领域的进一步应用.

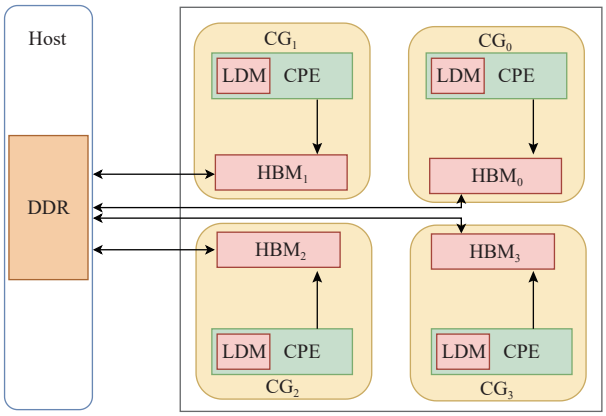

Fig. 5 Architecture of two-stier storage

图 5 两级存储架构

为充分利用片上环网总线的高级硬件特性,本文基于片上环网构建 L0(LDM)-L1(Local-HBM)-L2(Global-HBM)层次化三级存储架构. 具体而言,三级架构将 HBM 主存从物理和逻辑上划分为 2 个区域:一块作为全局内存(交叉段),可以通过环网直接进行读写操作,并由 4 个核组共享;另一块作为本地内存(连续区),仅供相应核组本地访问. 如图 6 所示,Local-HBM 对应环网配置的连续段,作为本地内存;Global-HBM 对应环网配置的交叉段,作为全局共享内存;LDM 性能最高,是每个从核的私有空间. 三级存储架构通过引入全局内存,提供了一个共享的数据交换平台,它允许核组之间通过全局内存进行高效的数据交换,核组仅需将交互的数据写入全局内存,显著降低了程序设计的复杂度,简化了数据交换的流程,减少了开发者在编写和调试程序时的工作量,提高了程序的可读性和可维护性,降低了因频繁通信带来的性能开销.

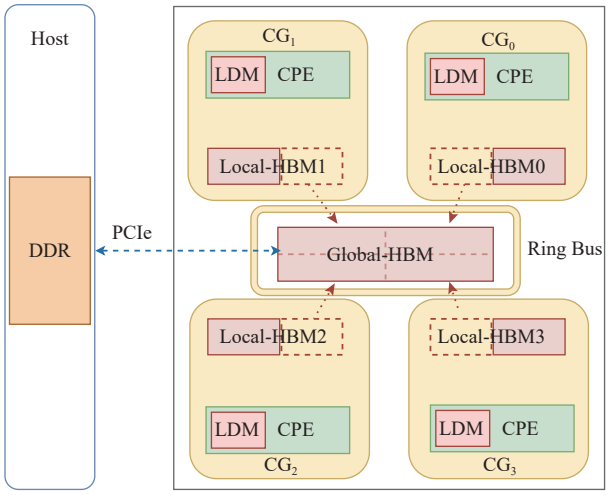

Fig. 6 Three-tier storage architecture based on ring-on-chip

图 6 基于片上环网的三级存储架构

三级存储架构能够显著提升单核组的显存容量. 在传统的二级架构中,每个核组的显存容量固定为 15 GB. 三级存储架构利用环网总线的交叉段特性,通过硬件的配置开关灵活地设置每个核组的本地 Local-HBM 和交叉共享 Global-HBM 空间的大小. Local-HBM 设置为 15 GB 时,Global-HBM 大小为 0; Local-HBM 设置为 0 时,Global-HBM 为 60 GB. 这种灵活性使得单核组可分配使用的最大显存容量达到传统方式的 4 倍,能够支持更大的数据集和批处理大小.

采用三级存储架构能够显著提高主机与加速卡之间的数据传输效率. 在传统的二级存储架构中,由于 4 个核组的本地内存 Local-HBM 是离散分布的,当主机与加速卡之间进行数据传输时,必须针对每个核组分别执行一次数据传输操作,如图 6 所示. 这种分散的数据传输方式不仅增加了传输次数,还可能导致 PCIe 传输带宽的浪费,进而降低了整体的数据传输效率. 相比之下,三级存储架构通过引入一块共享的全局显存来优化这一过程. 该架构下,训练推理任务中的模型参数仅需通过 1 次 sdaaMemcpy 操作就从主机拷贝数据到加速卡的全局显存中. 随后,4 个核组可通过高速环网共同复用这块全局显存中的模型参数进行计算,如图 7 所示. 由于减少了低带宽 PCIe 通路上不必要的重复传输,充分利用了片上环网高速传输,由 4 次 PCIe 传输变为 1 次 PCIe 传输加上 4 次环网传输,从而降低整体传输时间,提高了主机与加速卡之间的数据传输效率. 这种优化不仅提升了数据传输的速度,还减少了因频繁数据传输带来的系统开销,使得整个计算过程更加高效.

为了实现对三级存储的管理,结合 SDAA 运行时系统中原有内存管理算法,增加了对全局 HBM 的统一管理,三级架构内存分配原理如图 8 所示. 首先使用全局位图(bitmap)进行粗粒度分配,8 KB 粒度的空间支持主核页式访问,1 MB 大小的块是从核的段式访问空间,从核组-Local 空间对应三级存储中的 L1 部分,从核组-Global 空间为 L2. 在具体实现过程中结合 SW-AI 硬件特点,将位图存储区按照缓存行进行对齐,使用向量优化方法,提高了位图检索以及处理的速度. 然后使用 free_list 链表串起来的细粒度空间管理算法,内存块的头部存放该块内存的元数

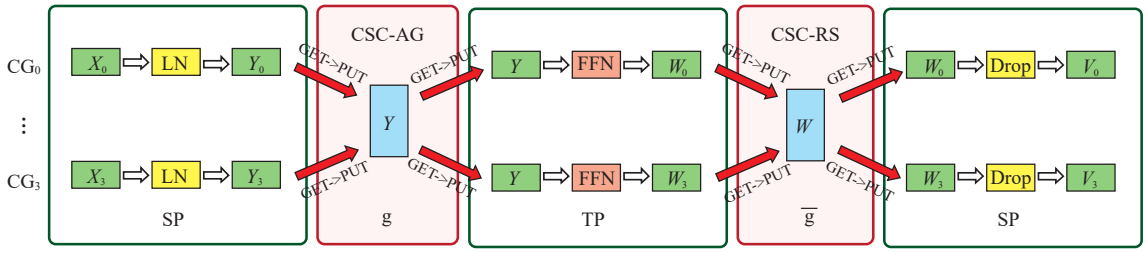

Fig. 7 SP+TP optimization based on CSC algorithm

图 7 基于 CSC 算法的 SP+TP 优化

据(meta data). 分配时, 从链表上分配一个对象出去; 释放时, 把对象插入到链表. 链表指针直接分配在待分配内存中, 不需要额外的内存开销. 该方法能够支持本地 HBM 段、交叉共享 HBM 段, 主核页内存统一管理, 具有分配回收速度快、数据块合并效率高、空间利用率高等优势. SDAA 运行时库中三级内存管理接口如表 1 所示.

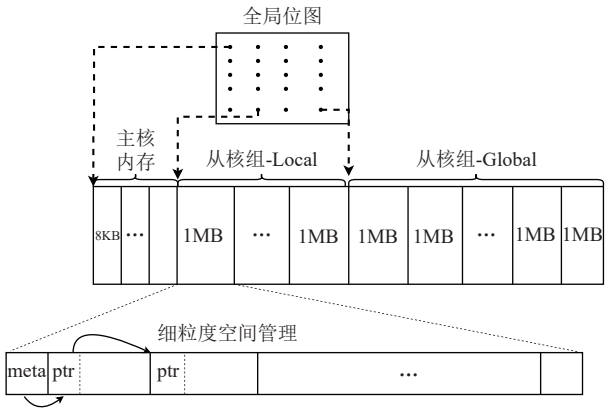

Fig. 8 Mechanism of memory allocation

图 8 内存分配机制

Table 1 Runtime Library Memory Management Interface

表 1 运行时库内存管理接口

SDAA 接口	功能描述	备注
sdaaMalloc	在设备端 Local-HBM 分配内存	已有接口，最大 15 GB
sdaaMallocCross	在设备端 Global-HBM 分配内存	新增接口，最大分配 60 GB
sdaaFree	释放 Local-HBM 已分配内存	已有接口
sdaaFreeCross	释放 Global-HBM 已分配内存	新增接口

2.2 采用交叉共享访存的片内通信算法

在申威智能加速卡的四核组架构中, 传统 Ring 算法的集合通信实现存在开销大、通信带宽低的问题. 在处理大模型训练推理等高内存需求任务时, 其通信效率不足严重制约系统整体性能表现.

为了解决上述问题, 本文基于申威智能加速卡的环网总线直接访问交叉共享内存的硬件特性与三级存储架构, 提出采用交叉共享访存的片内通信算法, 并设计实现卡内 TP+SP 通信最常用的通信算子 CSC-AG(cross share communication AllGather)和 CSC-RS(cross share communication ReduceScatter). 相较于传统 Ring 通信方式依赖 Send/Recv 操作的跨核组数据传输, CSC 算法的核心思想是利用片上环网总线支持直接访问交叉共享内存的硬件特性, 以访存操作访问全局内存 Global-HBM 的方式实现通信功能.

申威智能加速卡内四核组 CSC-AG 算法的实现与传统 Ring 通信算法有显著不同, 如图 9 所示, CSC-AG 传输模型相较于图 4 的 Ring-AG 算法具有三大优势: 1)主机 launch 次数从 12 降至 4, 同步次数由 3 降为 1, 降低 67% 的核函数启动和同步开销; 2)采用全并行读写架构, 突破 Ring 算法必需的 3 轮串行迭代限制, 实现端到端时延优化; 3)引入动态内存块调整机制, 通过 3 倍连续内存块读取策略提升带宽利用率, 有效缓解 Ring 算法固定块传输导致的带宽受限问题.

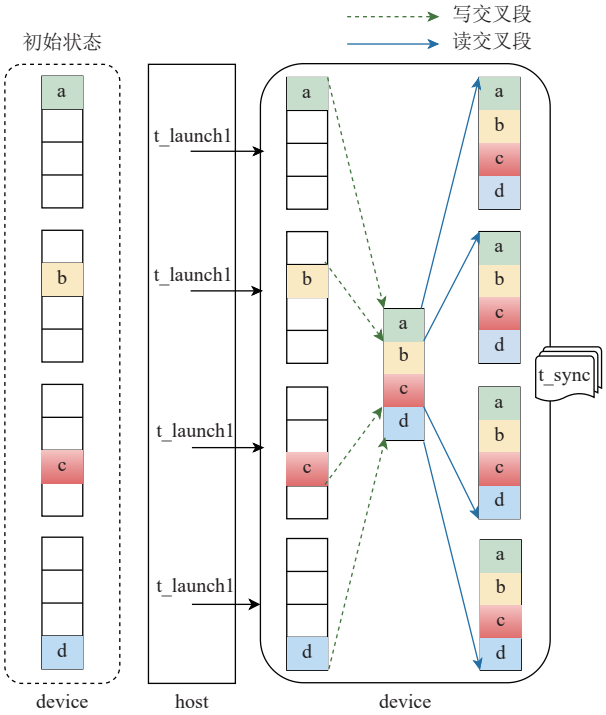

Fig. 9 CSC-AG algorithm in the four-core group

图 9 四核组 CSC-AG 算法

以单个核组上实现的 CSC 算法为例, 基于 CSC 算法实现的 All Gather 如算法 1 所示.

算法 1. CSC-AG.

输入: 每个核组输入数据 *input*, 核组号 *rank*, 总核组数 N=4;

输出: 每个核组包含来自所有核组的完整数据.

① CSC_AG(*input, rank, N*=4);

② *size* = *len*(*input*);

③ *hbm_global* = *hbm_local* = *rank* × *size*;

④ *DMA_GET*(*ldm*, *hbm_local*);

⑤ *DMA_PUT*(*ldm*, *hbm_global*);

⑥ *SYNC*();

⑦ FOR i = 1 TO N−1;

⑧ *hbm_global* = *hbm_local* = ((*i* + *rank*)MOD

N) × *size*;
⑨ *DMA_GET*(*ldm*, *hbm_local*);
⑩ *DMA_PUT*(*ldm*, *hbm_global*);
⑪ END FOR
⑫ END

算法 1 具体实现流程如下: 输入参数包括各核组输入数据 *input*、核组号 *rank* 及核组总数 *N*, 输出为包含全量数据的本地副本. 行①~⑥完成数据共享阶段, 各核组将待共享数据写入三级存储的交叉段后, 执行全局同步确保写入完成; 行⑦~⑪执行数据聚合阶段, 各核组并行读取交叉段中其他核组的数据.

算法 2. CSC-RS.

输入: 每个核组输入数据 *input*, 核组号 *rank*, 总核组数 *N*=4;

输出: 每个核组包含来自所有核组的 1/*N* 的 Reduce 数据.

① *CSC_AG*(*size*, *rank*, *N*=4);
② *size* = *len*(*input*);
③ *hbm_global* = *hbm_local* = *rank* × *size*;
④ *DMA_GET*(*ldm*, *hbm_local*);
⑤ *DMA_PUT*(*ldm*, *hbm_global*);
⑥ *SYNC*();
⑦ FOR *i* = 1 TO *N*−1;
⑧ *DMA_GET*(*ldm*, *hbm_global*);
⑨ *temp* = *Reduce*(*ldm*);
⑩ *DMA_PUT*(*ldm*, *hbm_global*);
⑪ END FOR
⑫ END

算法 2 展示了 CSC-RS 的实现框架, 其工作流程与 CSC-AG 保持结构一致性, 核心差异在于数据读取阶段需执行规约操作后写入本地.

以 AllGather 操作为例, CSC-AG 算法的带宽上限模型计算方法如式(4)所示.

$$BW_{\mathrm{AG}}=\frac{V_{\mathrm{AG}}}{t_{\mathrm{LAUNCH}}+t_{\mathrm{SYNC}}+3\times\left(\dfrac{V_{\mathrm{AG}}}{4\times BW_{\mathrm{GLOBAL}}}+t_{\mathrm{OTHERS}}\right)}. \tag{4}$$

带宽上限模型计算方法 BW_{AG} 是总数据量除以总耗时, 其中总耗时包括主机启动开销 t_{LAUNCH}、主机同步开销 t_{SYNC} 和内核执行开销等.

总数据量 V_{AG} 较小时(小于 1 MB), $\dfrac{V_{\mathrm{AG}}}{BW_{\mathrm{GLOBAL}}}\ll t_{\mathrm{LAUNCH}}+t_{\mathrm{SYNC}}+t_{\mathrm{OTHERS}}$, 式(3)(4)可以简化为:

$$BW_{\mathrm{AG-Ring}}\approx\frac{V_{\mathrm{AG}}}{3\times(t_{\mathrm{LAUNCH}}+t_{\mathrm{SYNC}}+t_{\mathrm{OTHERS}})}, \tag{5}$$

$$BW_{\mathrm{AG-CSC}}\approx\frac{V_{\mathrm{AG}}}{t_{\mathrm{LAUNCH}}+t_{\mathrm{SYNC}}+3\times t_{\mathrm{OTHERS}}}. \tag{6}$$

将式(5)(6)对比可知, 在数据量较小时, CSC-AG 算法理论带宽为传统 Ring 算法的 3 倍左右.

数据量 V_{AG} 较大时(大于 256 MB), $\dfrac{V_{\mathrm{AG}}}{BW_{\mathrm{GLOBAL}}}\gg t_{\mathrm{LAUNCH}}+t_{\mathrm{SYNC}}+t_{\mathrm{OTHERS}}$, 式(3)(4)可以简化为:

$$BW_{\mathrm{AG-Ring}}\approx\frac{V_{\mathrm{AG}}}{3\times\left(\dfrac{V_{\mathrm{AG}}}{4\times BW_{\mathrm{GLOBAL}}}+t_{\mathrm{OTHERS}}\right)}, \tag{7}$$

$$BW_{\mathrm{AG-CSC}}\approx\frac{V_{\mathrm{AG}}}{3\times\left(\dfrac{V_{\mathrm{AG}}}{4\times BW_{\mathrm{GLOBAL}}}+t_{\mathrm{OTHERS}}\right)}. \tag{8}$$

将式(7)(8)对比可知, 数据量较大时, CSC-AG 算法理论性能与传统 Ring 算法相当. ReduceScatter 算法性能与 AllGather 算法性能分析过程类似, 不再赘述.

与式(3)基于 Ring-AG 算法的带宽上限模型相比, 当单次 AllGather 传输量 V_{AG} 为 8 MB 时, 结合四核组并行启动特性, 其理论最大算法带宽可估算为 69.6 GB/s, 在相同数据规模下明显高于 Ring 算法 41 GB/s 的理论带宽.

在使用方式上, CSC 算法采用非侵入式设计, 可实现与 Ring-AG、Ring-RS 的无缝兼容替换, 无需修改现有应用架构即可直接部署. 将 CSC 算法用于优化图 3 所示的 Transformer 模型结构的序列并行和张量并行, 如图 7 所示. 在序列并行场景中, 各核组将输出数据 $[\boldsymbol{Y}_0,\boldsymbol{Y}_1,\boldsymbol{Y}_2,\boldsymbol{Y}_3]$ 写入 Local-HBM. CSC-AG 操作执行时, 首先完成 Global-HBM 写入及全局同步, 实现本地数据向交叉段分区的映射; 随后通过 Global-HBM 读取操作, 将增量数据并行加载至 Local-HBM, 最终达成全量数据同步. 每个核组获得全局数据 $[\boldsymbol{Y}]$.

2.3 双向算子融合的通信计算重叠方法

访存驱动的高效 CSC-AG, CSC-RS 通信算法充分利用了环网的硬件特性, 大大提升了带宽利用率. 但是 CSC-AG, CSC-RS 需要单独发起并等待完成, 通信操作与计算操作独立运行, 仍旧无法重叠. 这种串行化的执行方式导致通信延迟无法被有效掩盖, 限制了整体性能的提升.

CUDA(compute unified device architecture)编程模型利用 CUDA 流(stream)实现通信与计算重叠, CUDA 流是一个操作序列, 其中的操作按照提交顺序串行执行. 如果将通信操作任务分配到一个 CUDA 流, 将计算任务放入另一个独立的 CUDA 流中, GPU 中数据传输和计算依赖的部件相互独立, 通信和计算任务可在 2 条不同的流中并行执行, 这样就实现

通信操作和计算操作的并行化，掩盖了通信的延迟.

在申威智能加速卡上，难以直接通过流的方式实现通信与计算的重叠. 该加速卡的通信和计算都依赖从核发起 DMA 指令进行访存操作，两者共用从核及 DMA 引擎部件，即使将两者分配到不同的流上也无法真正并行. 从硬件资源角度来看，实际上通信操作主要占用环网带宽，并不占用从核的计算资源. 因此，在等待环网传输数据期间，从核的计算单元仍可以并行其他计算任务，从而实现硬件资源的优化利用.

为实现申威智能加速卡在 SP 和 TP 分布式训练场景中的高效通信与计算重叠，本文借鉴算子融合思想，结合三级存储架构，提出了一种双向算子融合的通信计算重叠方法. 图 10 展示了通信计算重叠优化后的 MLP 层次结构. 图中每一行表示一个核组的执行过程，标注的①②③分别对应 Layernorm[25]，GEMM（FFN 路径包含多个 GEMM 算子）以及 DROPOUT 算子的 3 个执行阶段：①从 Local-HBM 读取数据至局部高速存储区 LDM；②在 LDM 中执行计算操作，并将中间结果暂存于 LDM；③将最终结果写回 Local-HBM. 图中箭头表示数据流动方向，其中绿色箭头代表计算型算子，红色箭头代表 CSC 通信型算子，带虚线的白色箭头则表示被完全掩盖的操作.

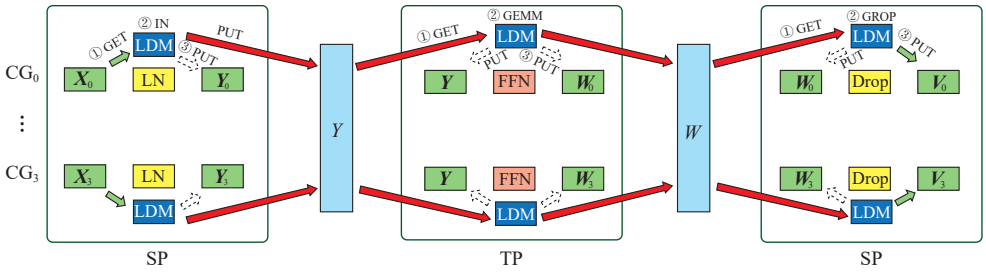

Fig. 10 Bidirectional operator fusion for overlapping methods in communication computing

图 10 双向算子融合的通信计算重叠方法

双向算子融合的通信计算重叠方法的核心思想是通过将通信算子与前序和后序计算算子融合，利用 LDM 绕过全局内存的中转环节，避免数据的重复存储与加载，从而实现通信与计算的高效重叠. 具体而言，该方法将 CSC-AG 与前序算子（LayerNorm）和后序算子（GEMM）融合，同时将 CSC-RS 与前序算子（GEMM）和后序算子（Dropout）融合. 通过这种方式，上一个算子的计算结果可直接从 LDM 传递至下一个算子，无需写入全局内存，显著减少了数据搬运开销.

以 0 号核组的 CSC-AG 为例，在 SP 阶段，LayerNorm 算子的执行分为 3 步：①从 Local-HBM 读取输入向量[$\boldsymbol{X}_n$]到 LDM；②在 LDM 上完成归一化计算；③将结果[$\boldsymbol{Y}_n$]同时写入 Global-HBM 和 Local-HBM（由于 Global-HBM 和 Local-HBM 位于不同路径，写操作可并行）. 这一过程融合了 CSC-AG 算子从 Local-HBM 搬运数据并聚合至 Global-HBM 的操作，节省了时间开销. 在 TP 阶段，数据[$\boldsymbol{Y}$]已经完整聚合至 Global-HBM，GEMM 算子第①步从 Global-HBM 加载数据[$\boldsymbol{Y}$]至 LDM，第②步在 LDM 上执行矩阵运算，同时并行执行 CSC-AG 算子将数据[$\boldsymbol{Y}$]写回 Local-HBM，从而实现通信与计算的重叠. 类似地，0 号核组的 CSC-RS 算子也通过相同机制实现了传输与计算的并行. 通过双向算子融合的操作，算子性能模型发生了变化，以下是算子在融合前和融合后数据传输的耗时对比：

$$T_1 = \frac{V_{\mathrm{AG}}}{BW_{\mathrm{LOCAL}}} + \frac{V_{\mathrm{AG}}}{BW_{\mathrm{GLOBAL}}} + \frac{3\times V_{\mathrm{AG}}}{BW_{\mathrm{GLOBAL}}} + \frac{4\times V_{\mathrm{AG}}}{BW_{\mathrm{LOCAL}}} + 3\times t_{\mathrm{LAUNCH}}, \tag{9}$$

$$T_2 = \frac{V_{\mathrm{AG}}}{BW_{\mathrm{GLOBAL}}} + \frac{3\times V_{\mathrm{AG}}}{BW_{\mathrm{GLOBAL}}} + \frac{V_{\mathrm{AG}}}{BW_{\mathrm{LOCAL}}} + 2\times t_{\mathrm{LAUNCH}}, \tag{10}$$

$$p = \frac{\dfrac{4\times V_{\mathrm{AG}}}{BW_{\mathrm{LOCAL}}}}{\dfrac{V_{\mathrm{AG}}}{BW_{\mathrm{LOCAL}}} + \dfrac{V_{\mathrm{AG}}}{BW_{\mathrm{GLOBAL}}} + \dfrac{3\times V_{\mathrm{AG}}}{BW_{\mathrm{GLOBAL}}} + \dfrac{4\times V_{\mathrm{AG}}}{BW_{\mathrm{LOCAL}}}}. \tag{11}$$

在式(9)~(11)中，T_1是融合前的耗时，T_2是融合后的耗时，p 是融合后的收益百分比. 其中t_{LAUNCH}表示主机启动开销，V_{AG}表示每个核组进行 Layernorm 运算的数据量，占 AllGather 通信数据的 1/4；BW_{LOCAL}表示本地 Local-HBM 的带宽；BW_{GLOBAL}表示经环网访问 Global-HBM 的带宽；T_1和T_2统计的时间由前序算子把结果写入内存、通信算子传输数据、后序算子从内存获取到数据 3 段构成. 经过计算，在 AllGather 通信数据量为 8 MB 的情况下，融合的收益大概为 50%.

图 11 从微观层次上展示了单个从核内的通信计算重叠的原理. LayerNorm 与 CSC-AG 交错运行，CSC-

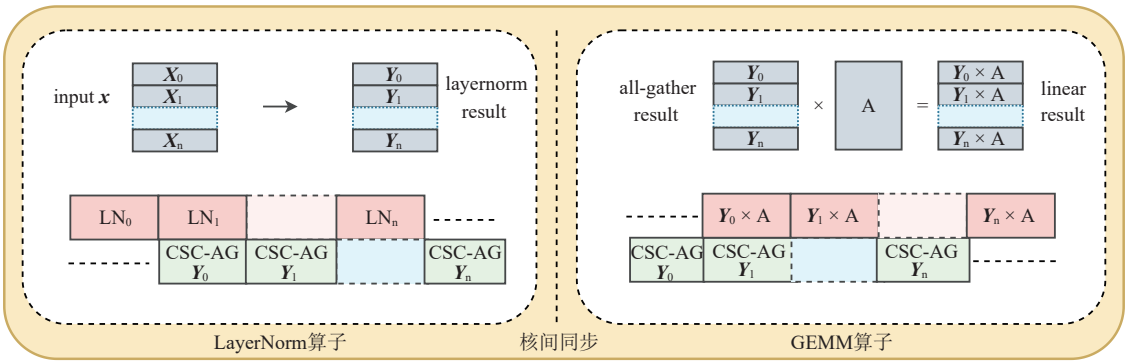

Fig. 11 Communication calculation overlapping of the slave core internal

图 11 从核内部通信计算重叠

AG 与 GEMM 交错运行, 前序算子 LayerNorm 在执行归一化运算的同时发起 CSC-AG 操作, 将运算结果 $\boldsymbol{Y}_n$ 写入到环网 Global-HBM; CSC-AG 在 Global-HBM 读取数据 $\boldsymbol{Y}_n$ 的同时, 后序算子 GEMM 并行执行. 由于 LayerNorm 和 GEMM 算子更大, 因此可以很好地掩盖掉 CSC-AG 的通信开销. 通过将通信操作与计算过程融合, 通信的延迟可以被计算任务掩盖, 实现高效的通信与计算重叠. 这种方式可以有效提高核组的使用率和整体性能, 尤其是在分布式训练中, 提升大规模分布式训练的效率, 从而确保大规模语言模型 LLM 训练的高效进行.

3 性能评估

实验评估主要针对基于三级存储架构设计的访存替代通信算法 CSC-AG 和 CSC-RS, 以及经过双向算子融合优化后的算法. 评估的目的是对比这些新算法与传统基于环网 Ring 方式实现的 AllGather 和 ReduceScatter 算法的性能差异.

测试硬件环境为一台型号为 NF5468M5 的浪潮英信服务器, 在其 PCIe 插槽上插入一块申威智能加速卡, 确保主机侧环境独占使用. 运行时测试环境如表 2 所示, AI-driver1.0 版本的环境对应的是传统二级存储架构下的软件, AI-driver1.0-T 版本的软件是使用三级存储架构下的运行时及驱动.

Table 2 Testing Environment

表 2 测试环境

服务器配置	操作系统	加速卡配置	驱动	运行时系统
CPU Silver4214 2.20 GHz 内存 DDR4 128 GB	CentOS 7.8	申威智能加速卡	AI-driver1.0-T	SDAA 1.0-T

3.1 主机-加速卡传输性能测试

本节对二级存储架构与三级存储架构下主机与加速卡之间通过 PCIe 传输数据的性能进行了对比分析. 实验通过选取不同规模的模型参数, 在主机与加速卡之间进行数据传输, 并测量其有效带宽. 在二级存储架构中, 测试方法为分别向 4 个 Local-HBM 各执行 1 次独立的 PCIe 传输; 而在三级存储架构中, 测试方法为先向 Global-HBM 执行 1 次 PCIe 传输, 随后通过 4 次环网传输将数据分发至各 Local-HBM. 图 12 展示了不同数据规模下 2 种存储架构的传输性能对比结果. 实验结果表明, 随着传输数据量的增加, 三级存储架构在传输性能上展现出显著优势, 其性能较二级存储架构提升了 2.5 倍. 由于大模型训练场景下数据传输以大规模数据为主, 三级存储架构能够更好地适应此类需求.

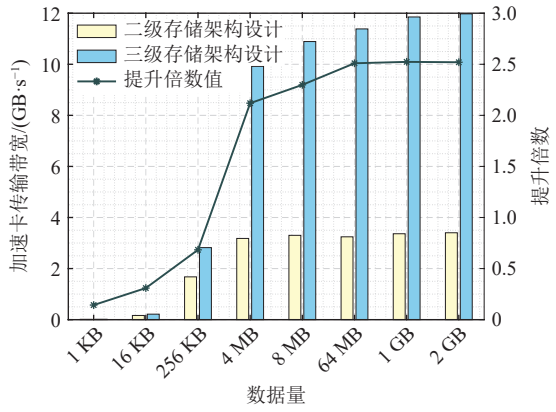

Fig. 12 Host-accelerator card transfer performance under different storage architectures

图 12 不同存储架构下主机-加速卡传输性能

3.2 CSC 算法性能测试

本节对基于二级存储架构设计的传统集合通信算法 Ring-AG 和 Ring-RS, 以及基于三级存储架构设计的交叉共享访存的片内通信算法 CSC-AG 和 CSC-RS 进行了性能评估. 图 13 展示了 Ring 和 CSC 算法的带宽利用率, 计算方法为数据传输时间占总时间

的比值. 由于 Ring 算法在主机启动和同步过程中存在较高的开销, 尤其是在小数据量情况下, 数据传输时间占总时间的比值仅为 20%, 带宽利用率较低. 相较之下, CSC 算法通过减少主机启动次数和同步次数, 提高了数据传输时间的占比, 从而实现了更高的带宽利用率. 随着数据量增大, CSC 和 Ring 算法的总体时间逐渐接近, 带宽利用率也趋于相似.

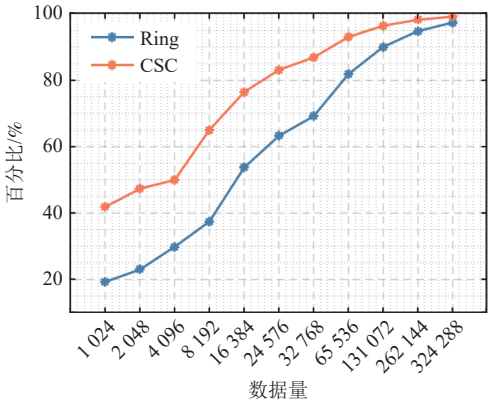

Fig. 13 Bandwidth utilization of CSC and Ring algorithms

图 13 CSC 算法和 Ring 算法的带宽利用率

图 14 展示了 CSC-AG 与 Ring-AG 算法的带宽对比结果. 实验表明, 在小数据量场景下, CSC-AG 的带宽显著高于 Ring-AG, 尤其在 4 096 KB 数据粒度时, CSC-AG 的性能达到 Ring-AG 的 2.15 倍. 这一性能优势主要源于 CSC-AG 优化了 Ring-AG 算法中因主机启动时间和同步时间占比较高而引入的开销. 随着数据量的增加, CSC-AG 与 Ring-AG 的带宽差距逐渐缩小, 两者性能趋于接近.

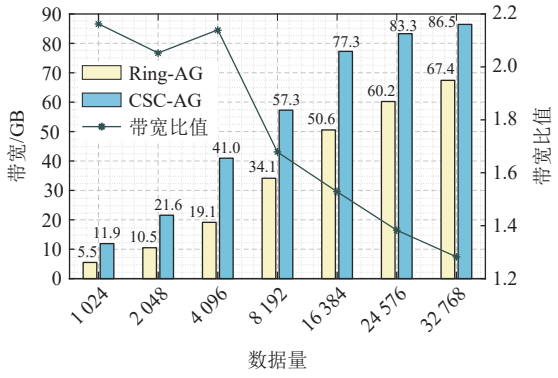

Fig. 14 Bandwidth comparison of CSC-AG and Ring-AG algorithms

图 14 CSC-AG and Ring-AG 算法的带宽对比

图 15 展示了 Ring-RS 和 CSC-RS 算法的带宽对比结果. 实验表明, 在小数据量场景下, CSC-RS 的带宽显著高于 Ring-RS, 在 32 768 KB 的数据粒度时, CSC-RS 的性能达到 Ring-RS 的 1.2 倍.

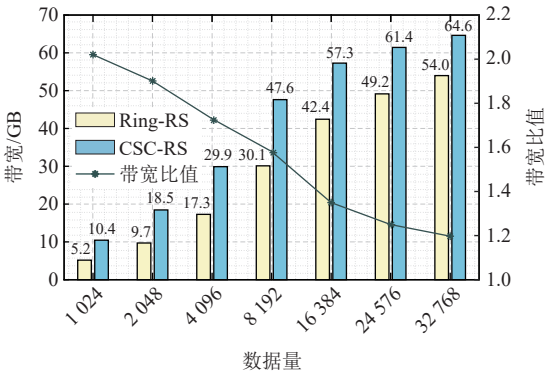

Fig. 15 Bandwidth comparison of Ring-RS 和 CSC-RS algorithms

图 15 Ring-RS 和 CSC-RS 算法的带宽对比

3.3 通信计算重叠测试

为验证双向算子融合的通信计算重叠方法的有效性, 本节以通信和计算算子的执行时间为核心指标来设计对照实验. 实验以 CSC-AG、CSC-RS 算子为测试对象, 选取不同的通信数据量进行性能测试. 表 3 和表 4 分别展示了 CSC-AG 和 CSC-RS 在应用双向算子融合的通信计算重叠方法后的性能收益. 在经过优化之前, 算子间互相独立, 中间结果需要通过内存中转; 在优化之后, 通信操作与计算操作融合, 中间结果直接通过 LDM 存取, 避免了冗余的内存访问开销. 实验结果表明, 在应用双向算子融合的通信计算重叠方法后, 通信算子的性能得到显著提升, CSC-AG 取得最大 59% 的性能收益, CSC-RS 取得最大 35%

Table 3 Experimental Data of CSC-AG for Communication Calculation Overlapping

表 3 CSC-AG 通信计算重叠实验数据

通信数据量/KB	通讯时间/μs	重叠收益/μs	收益百分比/%
2 048	32	19	59
4 096	55	29	52
8 192	200	46	54
16 384	195	84	43
32 768	356	181	50

Table 4 Experimental Data of CSC-RS for Communication Calculation Overlapping

表 4 CSC-RS 通信计算重叠实验数据

通信数据量/KB	通讯时间/μs	重叠收益/μs	收益百分比/%
2 048	54	19	35
4 096	98	32	33
8 192	184	58	32
16 384	347	116	33
32 768	631	209	33

的性能收益.

3.4 训练预期性能分析

本文研究目前已完成单卡原型验证, 为评估GPT-NeoX 20B, LLaMa 70B模型在多卡训练场景下的优化潜力, 基于实测数据构建性能预估模型如表5所示.

Table 5 Expected Performance of the Intra Card Communication Training

表5 卡内通信训练的预期性能

模型	通信方式	单层时间/s	每轮迭代/s	预训练/d
GPT-NeoX 20B	Ring-TP	1.28	3.52	6.1
	双向融合后 CSC-TP	0.80	2.20	3.8
LLaMa 70B	Ring-TP	1.15	5.75	34.9
	双向融合后 CSC-TP	0.71	3.56	21.6

表5中的GPT-NeoX 20B模型的训练数据为0.4 T tokens[26], LLaMa 70B模型的训练数据是1.2节的场景下的训练数据2 T tokens[27]. 表5中第1行表示使用传统Ring算法下卡内TP集合通信AG和RS对应消耗的总时间, 第2行表示使用双向融合后的CSC算法进行卡内通信对应的时间消耗. 从表5可以看到, 如果使用本文提出的双向融合CSC算法, 完成整个GPT-NeoX 20B模型训练预计总时间可以节约2.3天; 整个LLaMa 70B模型训练预计总时间可以节约13.3天.

4 总结展望

本文针对申威智能加速卡(SW-AI)在大模型推理任务中面临的显存容量及主机-设备带宽双重限制、卡内集合通信开销大、通信计算无法重叠等核心问题, 提出SW-IntraCC优化框架. 该框架通过构建基于片上环网的三级存储架构, 显著提升了单核组可用的显存空间, 并有效缓解了主机-加速卡数据传输竞争; 此外, 该框架设计并实现了一种采用交叉共享访存的片内通信算法, 该算法通过减少通信开销, 提升了CSC-AG与CSC-RS的带宽. 同时, 双向算子融合的通信计算重叠方法通过并行执行计算与通信任务, 降低了时间开销, 从而提升整体效率.

实验结果表明, SW-IntraCC在申威智能加速卡上的集合通信性能是传统Ring算法的2.15倍, 单核组显存容量最高扩大4倍, 主机-加速卡传输带宽提升2.5倍; 重叠优化后性能进一步提升59%. 这一成果不仅为申威智能加速卡在大模型计算环境下的应用提供了更高效的通信解决方案, 也为其他类似架构的加速卡提供了参考. 未来, 该研究将进一步丰富和完善SW-IntraCC接口, 增加在不同场景下的适应性和通用性, 以期早日把该研究工作尽快融入到SDAA生态中.

作者贡献声明: 赵玉龙、顾燕卿负责方案的整体设计并撰写论文; 田松涛、吴春志负责部分算法思路和实验方法; 汤凌韬、张鲁飞、秦晓军、刘鑫、陈左宁提出指导意见并修改论文.

参 考 文 献

[1] Wang Rui, Zhang Liuyang, Gao Zhiyong, et al. Progress in research on large models for edge-oriented intelligence[J/OL]. Computer Research and Development, [2025-02-07]. http://kns.cnki.net/kcms/detail/11.1777.TP.20250127.0921.006.html (in Chinese)
(王睿, 张留洋, 高志涌, 等. 面向边缘智能的大模型研究进展[J/OL]. 计算机研究与发展, [2025-02-07]. http://kns.cnki.net/kcms/detail/11.1777.TP.20250127.0921.006.html)

[2] DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning[J]. arXiv preprint, arXiv: 2501.12948, 2025. DOI: 10.48550/arXiv.2501.12948

[3] Ouyang S, Dong D Z, Xu Y M, et al. Communication optimization strategies for distributed deep neural network training: A survey[J]. Journal of Parallel and Distributed Computing, 2021, 149: 52–65. DOI: 10.1016/j.jpdc.2020.11.005

[4] Weingram A, Li Y, Qi H, et al. xCCL: A survey of industry-led collective communication libraries for deep learning[J]. Journal of Computer Science and Technology, 2023, 38: 166−195

[5] NVIDIA. NVIDIA DGX−1 System Architecture White Paper 2017[R/OL].[2024-12-30].https://images.nvidia.cn/content/pdf/dgx1-system-architecture-whitepaper1.pdf

[6] Tipparaju V, Nieplocha J, Panda D K. Fast collective operations using shared and remote memory access protocols on clusters[C]//Proc of Int Parallel and Distributed Processing Symposium. Washington, DC: IEEE Computer Society, 2003: 10

[7] Pabst S, Koch A, Straßer W. Fast and scalable CPU/GPU collision detection for rigid and deformable surfaces[J]. Computer Graphics Forum, 2010, 29(7): 2145−2154

[8] Xu Q, Jeon H, Annavaram M. Graph processing on GPUs: Where are the bottlenecks?[C]//Proc of 2014 IEEE Int Symp on Workload Characterization (IISWC). Piscataway, NJ: IEEE, 2014: 140−149

[9] Next Platform. The system bottleneck shifts to PCI-express[R/OL][2017-07-14]. https://www.nextplatform.com/2017/07/14/system-bottleneck-shifts-pci-express/

[10] Huang Tongtong, Chen Hao, Wu Chenfei, et al. Application of physical implementation scheme for Concurrent multi-die optimization[J]. Electronics Applications, 2023, 49(8): 30−35 (in Chinese)

（黄彤彤, 陈昊, 武辰飞, 等. Concurrent multi-die optimization 物理实现方案的应用[J]. 电子技术应用, 2023, 49(8): 30−35）

[11] Zeng Ronghao. Research on multi-target tracking and behavior analysis based on maritime[D]. Xi'an: Xi'an Electronic Science and Technology University, 2023. DOI:10.27389/d.cnki.gxadu.2023.001428 (in Chinese)
（曾嵘浩. 基于海上的的多目标跟踪与行为分析研究[D]. 西安: 西安电子科技大学, 2023. DOI: 10.27389/d.cnki.gxadu.2023.001428）

[12] Yu Minghui. Research on model transplantation and optimization based on domestic smart chip[D]. Hangzhou: Hangzhou University of Electronic Science and Technology, 2024. DOI:10.27075/d.cnki.ghzdc.2024.000968 (in Chinese)
（俞铭辉. 基于国产智能芯片的模型移植和优化的研究[D]. 杭州: 杭州电子科技大学, 2024. DOI: 10.27075/d.cnki.ghzdc.2024.000968）

[13] Zhao Yulong, Zhang Lufei, Xu Guochun, et al. SDAA: Runtime system for Sunway intelligent acceleration card[J]. Journal of Software, 2024, 35(12): 5710−5724 (in Chinese)
（赵玉龙, 张鲁飞, 许国春, 等. SDAA: 面向申威智能加速卡的运行时系统[J]. 软件学报, 2024, 35(12): 5710−5724）

[14] Wikipedia. RDMA over converged Ethernet[EB/OL]. Wikipedia, the free encyclopedia. 2017[2024-12-01]. https://en.wikipedia.org/w/index.php?title=RDMA_over_Converged_Ethernet&oldid=782744462

[15] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[C]//Proc of the 31st Conf on Neural Information Processing Systems (NIPS 2017). Long Beach, CA: Curran Associates Inc, 2017: 5998−6008

[16] Sergeev A, Balso M D. Horovod: Fast and easy distributed deep learning in TensorFlow[J]. arXiv preprint, arXiv: 1802.05799, 2018

[17] Shoeybi M, Patwary M, Puri R, et al. Megatron-LM: Training multi-billion parameter language models using model parallelism[J]. arXiv preprint, arXiv 1909.08053, 2019

[18] Huang Y, Cheng Y, Bapna A, et al. GPipe: Efficient training of giant neural networks using pipeline parallelism[C]//Advances in the 32nd Conf on Neural Information Processing Systems (NeurIPS 2019). Vancouver, CA: Curran Associates Inc 2019: 8−14

[19] Narayanan D, Harlap A, Phanishayee A, et al. PipeDream: Generalized pipeline parallelism for DNN training[C/OL]//Proc of the 27th ACM Symp on Operating Systems Principles. New York: ACM, [2025-01-15]. https://doi.org/10.48550/arXiv.1806.03377, 2019

[20] Li S, Xue F, Li Y, et al. Sequence parallelism: Long sequence training from system perspective[C]//Proc of the 61st Annual Meeting of the Association for Computational Linguistics. Toronto, Canada: 2021: 2391−2404

[21] Gao Yiqin, Luo Zhiyu, Wang Yichao, et al. Evaluation and optimization of operating system for domestic supercomputing[J/OL]. Computer Science, [2025-02-11]. http://kns.cnki.net/kcms/detail/50.1075.tp.20240925.1330.007.html (in Chinese)
（高亦沁, 罗智宇, 王一超, 等. 面向国产超算的操作系统评测与优化[J/OL]. 计算机科学, [2025-02-11]. http://kns.cnki.net/kcms/detail/50.1075.tp.20240925.1330.007.html）

[22] Li M, Andersen D G, Park J W, et al. Scaling distributed machine learning with the parameter server[C]//Proc of the 11th USENIX Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2014: 583−598

[23] Träff J L. Optimal, non-pipelined Reduce-scatter and Allreduce algorithms[J]. arXiv preprint, arXiv: 2410.14234, 2024

[24] Wasi-ur-Rahman M, Lu X, Islam N S, et al. HOMR: A hybrid approach to exploit maximum overlapping in MapReduce over high performance interconnects[C]//Proc of the Int Conf on Supercomputing (ICS). New York: ACM, 2014: 33−42. DOI: 10.1145/2597652.2597684

[25] Ba J, Kiros J R, Hinton G E. Layer normalization[J]. arXiv preprint, arXiv: 1607.06450, 2016

[26] Black S, Biderman S, Hallahan E, et al. GPT-NeoX−20B: An open-source autoregressive language model[J]. arXiv preprint, arXiv: 2204.06745, 2022

[27] Touvron H, Martin L, Stone K, et al. Llama 2: Open foundation and fine-tuned chat models[J]. arXiv preprint, arXiv: 2307.09288, 2023

Zhao Yulong, born in 1984. PhD candidate. His main research interest includes artificial intelligence infrastructure software.
赵玉龙, 1984 年生. 博士研究生. 主要研究方向为人工智能基础软件.

Gu Yanqing, born in 1992. Master. His main research interests include high-performance computing and operating systems.
顾燕卿, 1992 年生. 硕士. 主要研究方向为高性能计算、操作系统.

Tian Songtao, born in 1983. Bachelor. His main research interests include high-performance computing and operating systems.
田松涛, 1983 年生. 学士. 主要研究方向为高性能计算、操作系统.

Wu Chunzhi, born in 1991. PhD candidate. His main research interests include space launch equipment assurance and deep learning algorithms.
吴春志, 1991 年生. 博士研究生. 主要研究方向为航天发射装备保障、深度学习算法.

Tang Lingtao, born in 1994. PhD. His main research interests include machine learning, privacy protection, and intelligent operating systems.
汤凌韬, 1994 年生. 博士. 主要研究方向为机器学习、隐私保护、智能操作系统.

Zhang Lufei, born in 1986. PhD, senior engineer. His main research interests include high-performance computing, operating systems, and machine learning.

张鲁飞, 1986 年生. 博士, 高级工程师. 主要研究方向为高性能计算、操作系统、机器学习.

Qin Xiaojun, born in 1975. PhD. Full senior engineer. His main research interests include system security, software vulnerability analysis, and artificial intelligence.

秦晓军, 1975 年生. 博士. 正高级工程师. 主要研究方向为系统安全、软件脆弱性分析、人工智能.

Liu Xin, born in 1979. PhD, professor. Her main research interest includes parallel algorithms and applications.

刘　鑫, 1979 年生. 博士, 研究员. 主要研究方向为并行算法及其应用.

Chen Zuoning, born in 1957. PhD, PhD supervisor, Academician of Chinese Academy of Engineering. Her main research interests include software theory, operating systems, and information security.

陈左宁, 1957 年生. 博士, 博士生导师, 中国工程院院士. 主要研究方向为软件理论、操作系统、信息安全.