计算机研究与发展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202440055
61(6): 1388−1400, 2024

Direct xPU: 一种新型节点间通信优化的分布式异构计算架构

李仁刚[1,2] 王彦伟[2] 郝 锐[2] 肖麟阁[2] 杨 乐[3] 杨广文[1] 阚宏伟[3]
[1](清华大学计算机科学与技术系 北京 100084)
[2](浪潮(北京)电子信息产业有限公司 北京 100085)
[3](广东浪潮智慧计算技术有限公司 广州 510623)
(lirg@ieisystem.com)

Direct xPU: A Novel Distributed Heterogeneous Computing Architecture Optimized for Inter-node Communication Optimization

Li Rengang[1,2], Wang Yanwei[2], Hao Rui[2], Xiao Linge[2], Yang Le[3], Yang Guangwen[1], and Kan Hongwei[3]
[1](*Department of Computer Science and Technology*, *Tsinghua University*, *Beijing* 100084)
[2](*Inspur* (*Beijing*) *Electronic Information Industry Co.*, *Ltd*, *Beijing* 100085)
[3](*Guangdong Inspur Intelligent Computing Technology Co.*, *Ltd*, *Guangzhou* 510623)

Abstract The explosive growth of the application of large-scale artificial intelligence models has made it difficult to achieve the scale deployment of applications relying on a single node or a single type of computing architecture. Distributed heterogeneous computing has become the mainstream choice, and inter-node communication has become one of the main bottlenecks in the training or inference process of large models. Currently, there are still some deficiencies in the inter-node communicating solutions dominated by leading chip manufacturers. On the one hand, some architectures choose to use a simple but less scalable point-to-point transmission scheme in order to pursue the ultimate inter-node communication performance. On the other hand, traditional heterogeneous computing engines (such as GPUs) are independent of CPUs in terms of computing resources such as memory and computing cores, but they lack dedicated communicating network devices in terms of communication resources and need to rely entirely or partially on CPUs to handle transmission between heterogeneous computing engines and the shared communicating network device through physical links such as PCIe. The proposed Direct xPU distributed heterogeneous computing architecture in this article enables heterogeneous computing engines to have independent and dedicated devices in both computing and communication resources, achieving zero-copy data and further eliminating the energy consumption and latency associated with cross-chip data transfer during inter-node communication. Evaluations show that Direct xPU achieves communication latency comparable to computing architectures pursuing ultimate inter-node communication performance, with bandwidth close to the physical limit.

Key words inter-node communication; FPGA; GPU; RDMA; zero copy

摘 要 人工智能大模型应用的爆发式增长,使得难以依靠单一节点、单一类型的算力实现应用的规模部署,分布式异构计算成为主流选择,而节点间通信成为大模型训练或推理过程中的主要瓶颈之一.目前,主要由GPU,FPGA等头部芯片厂商所主导的各种计算架构的节点间通信方案还存在一些问题.一方面,为了追求极致的节点间通信性能,一部分架构选择使用协议简单而可扩展性差的点对点传输方案.另一

收稿日期: 2024-01-29; **修回日期**: 2024-03-27
基金项目: 广东省重点领域研发计划项目 (2021B0101400001)
This work was supported by the Key-Area Research and Development Program of Guangdong Province (2021B0101400001).
通信作者: 肖麟阁 (xiaolinge@ieisystem.com)

方面，传统的异构计算引擎（例如GPU）虽然在内存、计算管线等算力要素方面独立于CPU，但在通信要素方面却缺少专属的网络通信设备，需要完全或部分借助于CPU通过PCIe等物理链路来处理异构计算引擎与共享网络通信设备之间的通信. 所实现的Direct xPU分布式异构计算架构，使得异构计算引擎在算力要素和通信要素两方面均具有独立的、专属的设备，实现了数据的零拷贝，并进一步消除了节点间通信过程中处理跨芯片传输数据所带来的能耗和延迟. 测试结果表明，Direct xPU取得了与追求极致的节点间通信性能的计算架构相当的通信延迟，带宽接近物理通信带宽的上限.

关键词 节点间通信；FPGA；GPU；RDMA；零拷贝

中图法分类号 TP393

随着GPT等大模型的大量涌现和部署应用，同时具有计算密集型和I/O密集型特点的巨型神经网络使芯片算力和芯片间互联通信承压. 这一方面促进了数据中心和超级计算中心的融合；另一方面，也激发了底层架构的快速发展. 分布式异构计算是其中的研究重点之一. 高效的分布式计算不仅要求保持传统体系结构下高性能计算的优点，还要求提供与之匹配的高带宽、低延迟的节点间通信. 为了实现这一目标，学术界和工业界投入了大量的资金和精力，基于几种互连通信技术，如PCIe、RDMA等开发了各种编程范式和相应的硬件架构.

然而，现有的分布式异构计算体系结构似乎使得高性能、高带宽和低延迟成为一个不可能的三角[1]. 在这些分布式异构计算系统中，每个节点通过使用最先进的计算设备来实现高性能，而在节点之间，设备通过可选的几种不同的通信方式进行互联. 一般来说，高性能计算伴随着大量数据的生成和移动，带来数据重复拷贝开销、上下文切换、PCIe读写速度不均衡等一系列问题. 一个根本原因是在当前的分布式异构计算系统中，除了CPU之外的计算引擎在系统中并不是一个能完全独立实现计算和通信的角色. 例如，GPU之间的通信需要CPU的协助，或者GPU与CPU由于共享同一个网卡而被PCIe等物理链路限制带宽和拓扑结构.

我们的第1个想法是使所有计算引擎在分布式异构计算系统中具有完全的独立自主通信的能力，从而消除PCIe等物理链路的带宽和拓扑限制以及传统架构中由PCIe带来的读写速度不对称的问题. 我们的第2个想法是构建一套软硬件系统，以提供端到端的零拷贝能力. 此外，新型的计算架构与主流的编程范式兼容也是十分重要的一环.

本文的主要贡献包括3个方面：

1）提出了一种新型的分布式异构计算架构Direct xPU. 通过将分布式计算的2个关键要素——通信引擎层面和计算引擎芯片层面进行重新排布，实现更高效、更灵活的分布式异构计算系统.

2）基于FPGA构建了一个Direct xPU的原型系统，并实现了完整的软硬件基础设施，在有限的硬件条件下实现了GPU节点之间的类RDMA通信，延迟性能与之前的研究结果相当.

3）开发了xPU Box以及与之配套的一系列软硬件模块，以匹配数据中心的真实部署环境，充分挖掘基于Direct xPU新架构的新型系统拓扑结构，并基于原型系统进行了应用展示.

1 相关工作

登纳德缩放定律和摩尔定律的失效使得开发最先进芯片的边际效应愈发明显，引发了研究分布式异构计算系统的热潮. 在一个分布式异构系统中存在多个节点，每个节点中可能存在不同类型的计算引擎，通常包括CPU，GPU或FPGA. 如何实现跨节点通信以获得一条低延迟、高带宽的数据通路是分布式异构系统的一个关键问题. 因此，关于分布式异构计算系统中的节点间通信问题的研究主要集中在这3类计算引擎上.

NVIDIA作为GPU行业的领导者，在GPU节点间互联通信方面开发了GPUDirect系列技术，包括GPUDirect Shared Memory，GPUDirect P2P，GPUDirect RDMA和GPUDirect Async. 这些技术之间的主要区别如图1所示. 早期的GPUDirect Shared Memory将GPU内存中的原始数据复制到主机的页锁定内存中，主机CPU控制网卡从页锁定内存中读取数据并发送给远端的节点的NIC（network interface controller）[2]. 很明显，该过程包括多次数据移动，并且需要CPU的深度参与，从而导致数据传输延迟的增加，并占用不必要的CPU资源. GPUDirect P2P实现了数据的零拷贝，但仍然需要CPU的参与，并且由于PCIe物理链

路的限制, GPUDirect P2P 只支持同一节点内 GPU 之间的通信[3]. GPUDirect RDMA 通过使能同一节点的 GPU 和 NIC 之间的 PCIe 通信, 支持不同节点的 GPU 之间的通信, 但它也需要 CPU 触发 NIC 从 GPU 内存中读取数据, 数据从 GPU 内存传输至通信端口需要跨越多个芯片[4]. GPUDirect Async 则进一步实现了 GPU 自主地触发本节点中的 NIC[5]. 总之, 在 GPUDirect 系列技术中, 总是由单个节点中的多个计算引擎共享同一个网络通信设备, 这就意味着这些计算引擎与通信设备之间必定存在某些物理链路以提供直接或间接的触发路径, 在 GPUDirect 系列技术中, 由 PCIe 链路提供这样的路径. 此外, 数据从 GPU 内存至数据发送起点的路径中存在多次跨芯片数据传输.

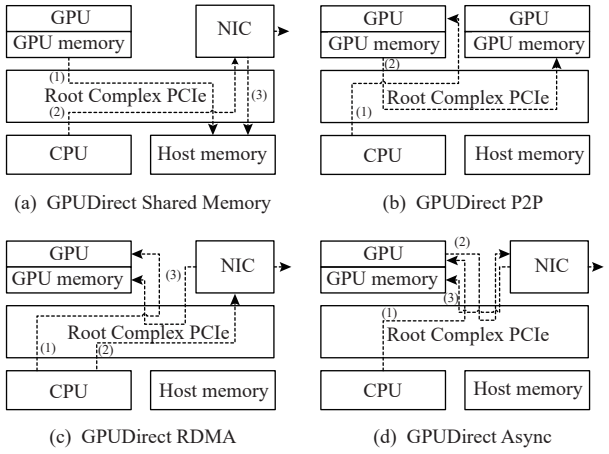


Fig. 1 Development of NVIDIA GPUDirect

图 1 NVIDIA GPUDirect 的发展

基于 NVIDIA 提供的软硬件, Oden 等人[6] 提出了一种名为 GGAS(global GPU address space)的通信模型, 该方法通过开发自定义的支持 RDMA 的 FPGA 网卡, 实现了 GPU 之间的零拷贝和无 GPU 协助的数据传输. 但同一节点内 CPU 和 GPU 仍共享一个通信设备, 其性能受限于 PCIe 链路. 此外, 由于当时普通芯片组的 PCIe 系统对 P2P 通信的支持较差, 导致实际带宽不到理论带宽的 1/3. 此外, Oden 等人[7] 还指出, 由于 GPU 的单线程性能与 CPU 有较大差距以及 PCIe 的开销, 使得 GPU 自主发送和接收网络流量的效率较低. Daoud 等人[8] 提出了 GPU 端 RDMA 通信库——GPUrdma, 实现了 GPU kernel 间的高效通信. 然而该方法需要使用专用的 PCIe Switch 以避免读写速度的显著差异, 并且由于 NVIDIA GPU 的硬件限制和闭源性, 需要花费大量精力修改相应的 GPU 和网卡驱动程序. Silberstein 等人[9] 为 GPU 的 kernel 程序提出了一种名为 GPUnet 的网络抽象模型, 该抽象模型提供了一个类 socket 的 API, 以旁路 CPU 实现 GPU 的自主通信, 然而 NVIDIA GPU 的局限性使得其不可能完全旁路 CPU, GPUnet 实际上仍需要 CPU 的协助来处理网络数据包, 例如 GPU 和 CPU 之间的流控同步等. 总之, 由于 PCIe 链路和共享同一个网卡的限制, 现有的方案难以实现有效的 GPU 独立自主通信.

FPGA 厂商和研究人员也纷纷涌入分布式异构领域. Intel 发布了用于分布式 FPGA 间直接通信的 IKL(inter-kernel links)[10], 如图 2 所示. IKL 使得 FPGA 在无需 PCIe 链路和 CPU 参与的情况下能够进行直接通信, 因为该类型的 FPGA 具有非共享的网络通信模块. 然而, 这种基于信用的流量控制和基于信道的 P2P 通信模型的复杂性极大地削弱了系统的可扩展性和互联性. IKL 的另一个问题是该技术仅适用于 FPGA 之间的通信, 而不支持 CPU 与 FPGA 之间的数据传输.

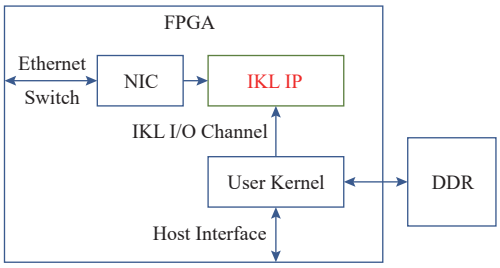


Fig. 2 Illustration of Intel IKL

图 2 Intel IKL 示意图

Fujita 等人[11] 实现了一项与 IKL 类似工作, 名为 CoE(channel over Ethernet), 该方案中使用第三方的以太网控制器模块来实现 IKL IP 的功能, 随后他们进一步提出了一种通信集成可重构计算系统(communication integrated reconfigurable computing system, CIRCUS)[12], 该系统基于 P2P 拓扑和 SeriaLite III 协议代替交换机拓扑和以太网协议, 以提高带宽和降低延迟. CoE 和 CIRCUS 之间的权衡实际上是效率和兼容性之间的竞争. Kobayashi 等人[13-14] 提出了 AiS (Accelerator-in-Switch), 以实现 GPU 和 FPGA 之间或 FPGA 之间的直接通信, 但是在该方案中, GPU 之间的数据传输仍需要进行多次数据拷贝, 并且 PCIe 链路所带来的带宽和系统拓扑限制仍然是一个问题. 更进一步, DPU 作为具有独立组网功能的领域专用加速器的代表, 越来越受到 NVIDIA, Intel 等公司[15-16] 的重视, 为了实现 RDMA 网络数据包处理、NVMe 存储访问加速、AES/RSA 加解密加速等复杂功能, 这些 DPU 通常不仅包含了 FPGA 以及 DRAM

等外部设备，还包含了 ARM/X86 CPU 等通用处理器，为上述复杂功能提供支撑，其本质上是以另一种形式构建了一个完整的通用系统，不可避免地引入了高昂的额外成本. Lant 等人[17] 针对 FPGA 类型的异构加速器提出了与本文类似的观点，即 FPGA 加速器应该是分布式异构系统中的完全对等的个体，他们开发了自定义互连解决方案，该方案将 CPU、FPGA 异构加速器和网络协议栈集成到单个 FPGA 芯片中，使 CPU 和 FPGA 计算引擎均可以完全独立控制网络设备. 尽管该方案没有解决 CPU 与其他计算引擎位于不同芯片的情况下各计算引擎如何独立控制网络设备的问题，但该方案中所展示出来的改进仍具有一定的参考意义.

综上所述，对于不具有独立通信组件的 GPU 的分布式节点间通信，由于需要通过 PCIe 与 CPU 共享同一个网络设备，使得 GPU 不可能完全旁路 CPU 实现独立自主通信，带来了额外的延迟，也限制了 GPU 设备的网络拓扑灵活度. 对于通过 FPGA/ASIC 实现的类 DPU 设备，一方面，其追求极致的节点间通信性能，极大地牺牲了系统的可扩展性；另一方面，通过在 DPU 中集成额外的 CPU 追求完全的通用性和功能性带来了额外的成本.

2 Direct xPU 概述

2.1 背 景

RDMA 在高速网络通信领域发挥着关键作用，并逐渐成为分布式计算的底层标准硬件设施之一[18-19]. NVIDIA 首先在 GPUDirect 系列技术中引入了 RDMA 技术，即 GPUDirect RDMA，用于减少设备之间移动数据时的拷贝数量. 一个典型的场景是，网卡可以通过 GPUDirect RDMA 直接访问 GPU 的内存，而不需要将数据复制到主机内存中，尽管需要 CPU 通过 PCIe 向网卡发送命令进行控制或同步. 目前已有多种 RDMA 实现，包括 InfiniBand, iWARP, RoCE 等，如表 1 所示. 由于 RoCE 协议的兼容性较好，RoCE

Table 1 Features Comparison of Three Types of RDMA Implementations

表 1 3 种 RDMA 实现的特性对比

特性	InfiniBand	iWARP	RoCE
性能	最好	稍差	与 InfiniBand 相当
成本	高	中	低
稳定性	好	差	较好
交换机需求	IB 交换机	以太网交换机	以太网交换机

RDMA 实现成为主流解决方案之一，因此本文的 RDMA 通信也采用了类似的实现方案.

OpenCL 是一个受大部分加速设备类型支持的开源编程框架，已成为异构编程的常见标准之一. OpenCL 规范中的 API 通常分为主机 API(通常在 CPU 上执行)和设备 API(通常在 GPU 或其他加速器上执行). OpenCL 使程序员可以完全控制加速器的存储层次和工作负载粒度. 为了提高多种类型支持 OpenCL 的设备的互操作性，Jääskeläinen 等人[20] 提出了可移植的 OpenCL——PoCL，即 OpenCL 规范的一个开源实现. PoCL 使用 Clang 作为 OpenCL 的编译器前端、LLVM 作为编译器后端. 在本文分布式异构计算系统中，我们采用 OpenCL 作为编程语言，实现了从其他平台的无缝迁移. 此外我们为本系统开发并集成了一套类 OpenCL 的 API，实现主机启动任意 GPU 节点之间的类 RDMA 通信的功能.

Vortex 是一个基于 RISC-V 的开源 GPGPU，它通过扩展 RISC-V 的指令集实现了 GPU 的计算功能. 并且支持 OpenCL 和 OpenGL 规范[21-22]. 在本文系统中，xPU 可以是任意类型的加速器. 由于 GPU 的普遍适用性以及 RISC-V 的开源特性，本文采用了 Vortex GPU 作为 xPU 的一个实例，后续表述将使用 Vortex GPU 作为 xPU 的代表性案例.

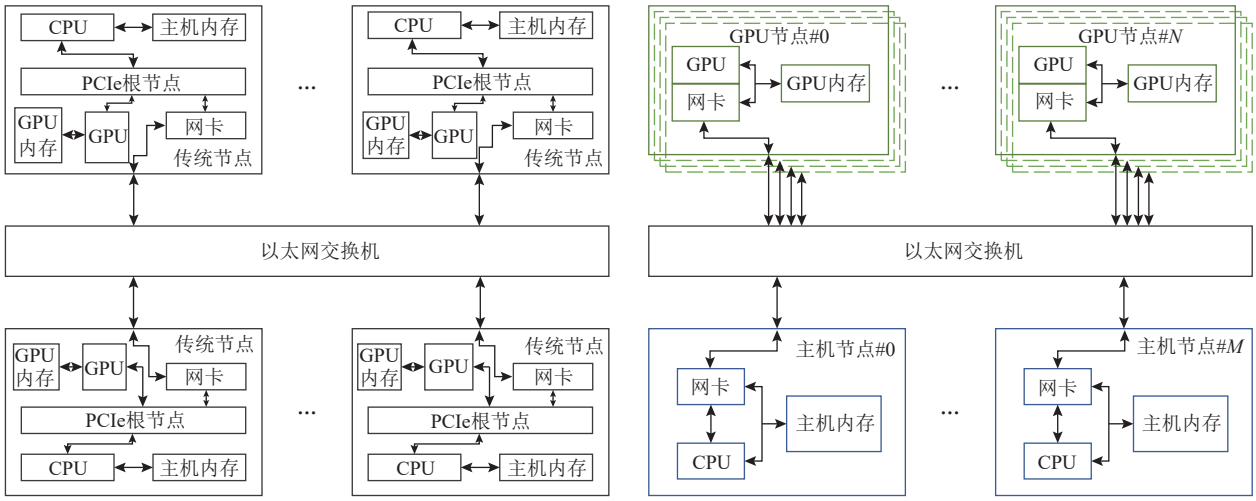
2.2 系统组织架构概述

基于 Direct xPU 的系统组织架构如图 3(b)所示. 在主机节点上，CPU 和网卡通过 PCIe 连接. 在 GPU 节点上，位于同一芯片中的 GPU 和网卡通过片上总线连接. 与图 3(a)所示的传统组织架构相比，我们在新体系结构中实现了一些明显的优势：1)系统中的任何节点都有一个专属的、服务于该节点内的单个计算引擎的网卡，使得每个计算引擎成为算力网络中完全独立自主的个体. 2)GPU 节点还具有降低通信延迟、减少功耗的天然优势，即通过计算设备 GPU 与通信设备网卡紧密结合，消除了 PCIe 链路绑定，并为计算引擎和网络设备之间高效协作提供了可能. 3)通过为各网卡实现 RDMA 功能，我们可以自然地获得低延迟、高带宽的数据通路. GPU 节点间的通信步骤为:

1)CPU 准备通信任务，并通过主机端的网卡向 GPU 发送通信任务.

2)GPU 根据通信任务的信息，控制 GPU 节点的网卡直接从 GPU 内存中读取数据.

3)GPU 节点的网卡向远端 GPU 节点发送数据.



(a) 传统GPU (b) Direct xPU

Fig. 3 Illustrations of systematic organization architecture based on the traditional GPU and Direct xPU

图 3 基于传统 GPU 和 Direct xPU 的系统组织架构示意图

3 Direct xPU 硬件架构

由于面向通用性服务的主机节点的硬件架构目前已经趋于成熟，因此关于主机节点，本文的工作主要集中在软件系统方面，而硬件架构主要描述了 GPU 节点的设计和改进. Direct xPU 的硬件架构如图 4 所示，其使用了 FPGA 作为原型系统的开发平台.

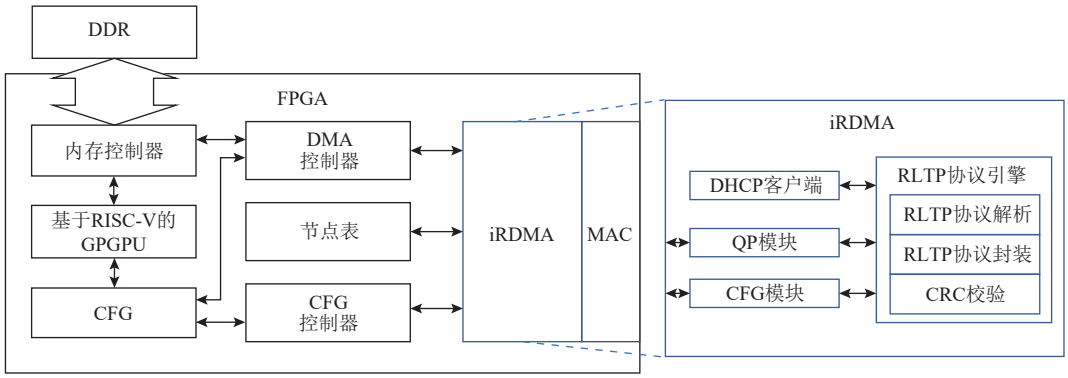


Fig. 4 Illustrations of architecture of Direct xPU hardware

图 4 Direct xPU 的硬件架构示意图

节点间通信时，实现了可靠的轻量级传输协议的 iRDMA 模块负责解析来自于 MAC 的数据包，根据数据包的功能，将处理后的数据包流转至内部的 DHCP 客户端模块(主要负责节点在系统中的初始化)、QP 模块(主要负责节点与其他节点间的数据交换)或者 CFG 模块(主要负责查询和控制 GPU 的状态). DMA 控制器根据 QP 模块的信号，与 DDR 内存进行数据交换. CFG 控制器根据 iRDMA 中 CFG 模块的信号，实现了对与 GPU 相连的 CFG 中的各类寄存器的访存，以获取和控制 GPU 的状态，此外，GPU 通过执行特定的程序指令，也可以访存与之相连的 CFG 中的各类寄存器，实现与 DMA 控制器的交互.

3.1 可靠的轻量级传输协议

为了加强 Direct xPU 与当前普通网络的兼容性，我们需要为 Direct xPU 实现与 RoCEv2 RDMA 类似的功能. 此外，为了建立一个可靠的链接，通常的做法是使用 TCP 协议，然而，与 UDP 协议相比，高度复杂的 TCP 协议将会消耗大量的 FPGA 硬件资源，并在数据路径中引入更多的延迟. 为此，在网络协议栈的传输层，我们定义了一种基于 UDP 的可靠的轻量级传输协议(Reliable Lightweight Transport Protocol, RLTP)，如图 5 所示. RLTP 提供超时/乱序重传、流量控制和拥塞管理等基础功能. 由于 RLTP 依赖于 UDP，不需要像 TCP 那样为每条链路的接收和发送创建专用的缓冲区，大大降低了对 FPGA 资源的需求和设计的复

Fig. 5 Illustration of RLTP protocol

图 5 RLTP 协议的示意图

杂性. 基于 RLTP 的网络数据包能实现多种功能, 包括节点间的正常数据交换, 以及传递控制或管理网络上的各类设备的消息. 利用 RLTP 协议中的“帧类型”字段区分数据包的功能, 使不同功能的数据包被交付给配置(Config, CFG)模块或 QP(Queue Pair)模块进一步处理. “目的 ID”或“源 ID”字段对应于网络上每个设备的唯一 ID, 以便快速根据节点表查找该 ID 对应的节点在网络中的 IP 地址. 通过 RLTP over UDP 方法, 可以在不涉及 CPU 的情况下, 将设备之间直接通信所需的基本网络协议栈完全卸载到 GPU 端, 该方法实际上是在有限的硬件资源条件、链路的可靠性和协议的轻量化之间的一种权衡.

3.2 基于 RISC-V 的 GPGPU 及其配置模块

系统在启动或运行过程中需要提供一些不可缺少的状态信息, 并提供更改这些状态信息的方式. 为了简化这一过程, 我们将这些零散信息和控制方式组织成一个统一的 CFG 模块. CFG 由一些控制寄存器和状态寄存器组成, 可以通过 GPU 或 iRDMA 模块访问. CFG 模块提供了一种获取整个系统关键信息以及将 GPU 的通信操作传输到 DMA 控制器的途径.

原生的 Vortex GPU 通过 PCIe 与主机进行通信, 软件方面则依赖于 Intel OPAE 接口[23], 这不利于 GPU 在通信网络上作为完全独立的对等体的实现. 因此, 我们只使用其计算核心, 该计算核心通过 DDR 控制器访问 GPU 内存. 此外, 通过在 CFG 模块中添加一系列可通过网络数据包访问的控制状态寄存器, 实现网络对 GPU 的使用和控制. 使用 Vortex GPU 的主要目的不是获得一个可与商用 GPU 相媲美的高性能 GPU 计算引擎, 而是实现和验证所提出的理论框架的可行性, 因此 GPU 性能不是本文的主要关注点.

3.3 类 RDMA 传输: iRDMA

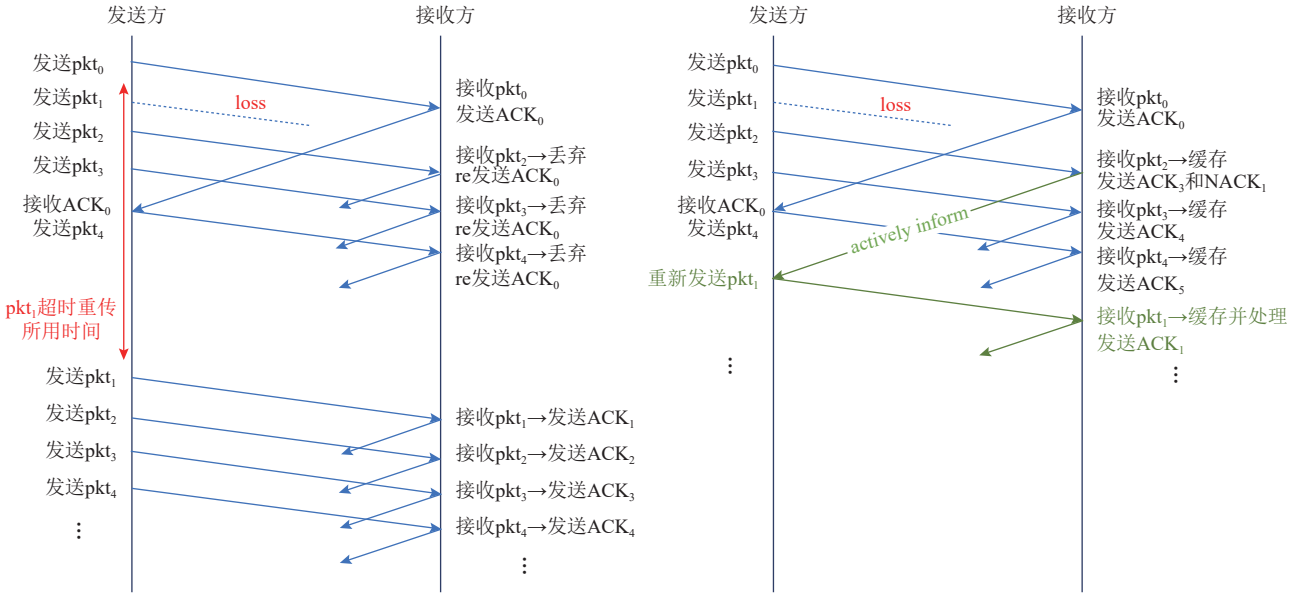
iRDMA 负责处理来自网络的、经过 RTLT 协议封装后的数据包. 首先, DHCP 客户端与上层的驱动程序配合工作, 在网络环境中注册该 GPU 节点, 使其在网络中可用, 并自动感知网络拓扑; 其次, 主机节点可以通过在 RLTP 头中发送具有正确字段的数据包来读写 GPU 节点内存的数据或者控制状态寄存器; iRDMA 将预处理后的数据包分配给 QP 模块或 CFG 模块, 分别负责完成对 GPU 内存/控制状态寄存器的读写操作. 再次, 主机节点的 CPU 下发 GPU 节点之间的通信任务时, 将通信操作所需的基本信息写入到 CFG 模块中. 最后, GPU 节点根据该信息执行相应的 RDMA 操作, 即从 GPU 内存中取出要发送的数据, iRDMA 模块将封装并发送数据包到远程 GPU 节点. 由于目前没有充分利用 GPU 和网卡之间的紧耦合性, GPU 之间的通信表现出的一系列行为并不完美. RISC-V 的指令集具有高度的模块化和可定制性, 未来考虑通过自定义指令直接向 CFG 模块提供 RDMA 操作所需的信息.

建立一个无损、可靠的网络链接的关键之一是实现高效的丢包重传. 在 IB RDMA 网络中, 该重传机制基于 Go-back-N 实现, 由于 IB RDMA 网络的丢包率接近于零, 采用这种策略对网络性能的影响可以忽略不计. 标准的、基于 UDP 的 RoCE RDMA 网络继承了这种策略. 然而, 与 TCP 协议相比, UDP 协议缺乏滑动窗口和确认应答等机制来实现可靠传输[24]. 一旦发生丢包, 需要依靠上层应用的检测来实现重传, 大大降低 RDMA 的传输效率、增加通信时延. 类似地, 如果我们不做改变, 基于 UDP 的 iRDMA 也会受到该缺点的影响.

具体来说, Go-back-N 策略意味着, 如果一个数据包未能正确到达接收方, 接收方将丢弃所有后续的数据包而不通知发送方. 而发送方只有检测到超时后, 才会重新发送该丢失数据包之后的所有数据包.

在 RLTP 报头的“子包编号”和“标志位”等字段, 我们基于 IRN 实现了选择性重传策略[18], 该策略使用位图来跟踪数据包. 当发生数据包丢失时, 接收端根据配置持续缓存一定范围内的后续分组, 并通过发送带有附加 NACK 字段的响应主动通知发送方启动选择性重传. 图 6 显示了在有序传输情况下, Go-back-N 与选择性重传的差异. 在 Go-back-N 策略中, 发送方向接收方依次发送了 pkt_0~pkt_4 数据包, 由于某种原因, pkt_1 数据包未能到达接收方, 接收方会根据报头中的信息丢弃已收到但顺序不正确的数据包, 同时反复发送 pkt_0 的确认包 ACK_0; 发送方无法接收

到 pkt_1 对应的应答包 ACK_1，超时之后，发送方会重传 pkt_1 及其后续所有的数据包. 而在选择性重传策略下，当 pkt_1 数据包未能到达接收方时，接收方会缓存后续顺序不正确的数据包，发送对应的 ACK 数据包，并通过发送 $NACK_1$ 主动通知发送方，以重传丢失的 pkt_1；当接收方收到发送方重传的 pkt_1 数据包时，则从缓存中取出所有数据包 pkt_2~pkt_4，进行后续处理. 实际场景中也存在乱序传输，但基本原理是相同的.



(a) Go-back-N策略 (b) 选择性重传策略

Fig. 6 Go-back-N strategy and the selective repeat transmission strategy

图 6 Go-back-N 策略和选择性重传策略

我们总结了 3 种 RDMA 实现的一些特性，如表 2 所示的丢包重传. 根据各自协议的标准，可以看出 iRDMA 在系统的可靠性、重传效率和实现难度之间取得了合理的平衡.

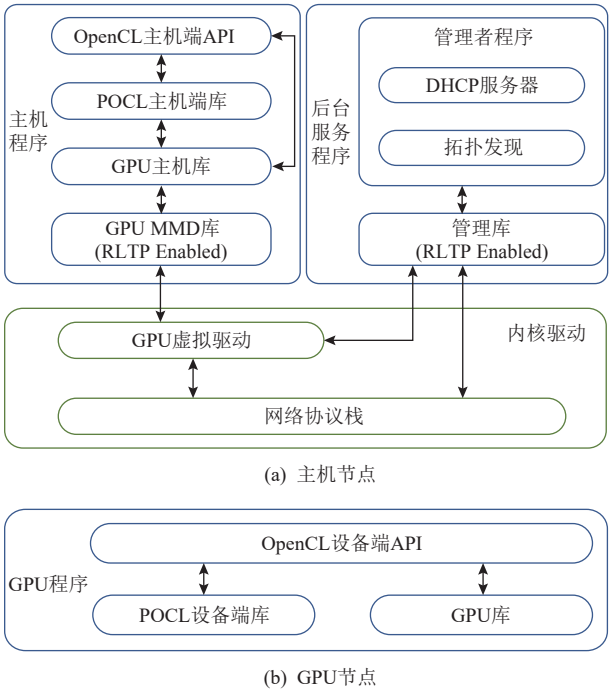
Table 2 Features of Three Types of RDMA Implementations in Terms of Retransmission

表 2 3 种 RDMA 实现在重传方面的特性

网络类型	传输层	重传协议	复杂度	可靠性
iWARP	TCP	SR	高	高
RoCEv2	UDP	Go-back-N	低	低
iRDMA	UDP	SR	中等	与 iWARP 相当

4 Direct xPU 软件架构

图 7 为 Direct xPU 系统的软件架构示意图. 在基于 Direct xPU 架构的分布式异构系统中，由于每个 GPU 节点可以作为网络上完全独立的通信节点，而不是僵化地由某个特定的主机节点所拥有，有利于大规模敏捷部署和资源池化. 为此，我们设计并开发了一个有效的软件基础设施，使 GPU 节点具有网络自动拓扑发现和管理等必要功能. 各层级的软件模块的设计遵循低耦合、高内聚的原则，逐层封装强硬件相关的实现细节，极大地提高了系统的可移植性



(a) 主机节点

(b) GPU节点

Fig. 7 The software architecture of Direct xPU

图 7 Direct xPU 的软件架构

和灵活性.

根据模块的功能和所服务的对象，软件基础架构主要由 4 个部分组成：后台服务软件库、GPU 虚拟驱动、主机应用程序库和 GPU 应用程序库.

4.1 后台服务软件库

正如 3.3 节所述，当 GPU 节点启动并初始化时，它通过 DHCP 客户端完成对自身的注册，并从后台服务程序中的 DHCP 服务器获得自己的 IP 地址. 此外，后台服务程序也是具有唯一性的 GPU 节点管理器，通过发送或解析 RLTP 数据包，在已注册的 GPU 节点之间执行一些管理和同步任务，如为每个 GPU 分配一个唯一的 ID，周期性或主动地将所有 GPU 节点的信息同步到网络中所有节点的节点表中，包括 IP 地址、mac 地址和设备 ID 等. 为了增强 GPU 节点资源池的灵活性，还支持手动指定特定主机节点可以发现和使用的特定 GPU 节点.

4.2 GPU 虚拟驱动

GPU 虚拟驱动是运行在内核态的底层模块，主要提供 2 个基本功能. 一是将 GPU 作为网络设备在 Linux 系统中进行枚举，以实现主机节点对 GPU 节点的远程访问；当某个主机节点通过 GPU 虚拟驱动从后台服务程序中获取某些 GPU 节点的必要信息后，将利用 RLTP 协议枚举并初始化对应的 GPU 节点. 二是向上层软件库——GPU MMD（memory mapped device）提供访问底层硬件的软件接口，以抽象主机节点与 GPU 节点之间的 RDMA 直接通信操作，生成或传递对应的 RTLT 数据包交付给网络协议栈.

4.3 主机应用程序库

Vortex GPU 团队引入并修改了 PoCL，使 Vortex GPU 具有良好的可移植性，这是本文选择 Vortex 作为 xPU 原型组件的重要原因. 对于主机节点的主机端程序来说，PoCL 主机端库负责将 OpenCL 主机 API 转换为主机节点对 GPU 节点的各种操作，即 GPU 主机库. GPU 主机库的 API 函数体使用了 Intel OPAE 接口，在本系统中使用自定义的 MMD 库替代基于 PCIe 的 OPAE 接口，以实现对 GPU 的远程调用，并向上保持接口的一致性. 此外，为了向终端用户提供控制 GPU 节点间通信的方法，在 GPU 主机库中添加了类 OpenCL 的 APIs，该 APIs 通过直接调用 GPU MMD 库实现主机下发 GPU 节点间通信任务的操作.

图 8 进一步说明了 GPU MMD 库在软件栈中的关键作用. GPU MMD 库向 GPU 主机库提供了一组更细粒度的操作——GPU 的 API. GPU MMD 库工作在 GPU 虚拟驱动程序之上，公开了与 GPU 虚拟驱动程序直接交互的接口. GPU 虚拟驱动程序实现了将远端的 GPU 节点枚举为本地的 GPU 设备的功能. 这种将远端 GPU 节点“虚拟”地暴露为同一节点中本地 GPU 设备的方法，为其他基于传统的 PCIe 链路的 xPU 的移植或开发提供了极大的便利性和可行性. 总而言之，选择这种实现方式一方面是移植 Vortex GPU 到原型系统中的需求，而另一方面则是松耦合的软件分层设计为 Direct xPU 带来了灵活和广阔的设计空间.

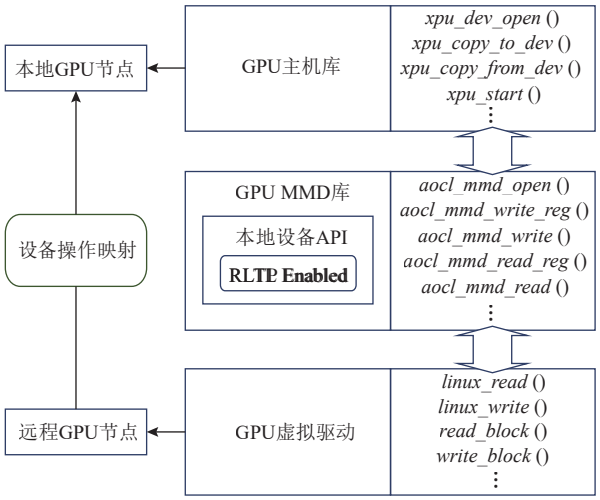


Fig. 8 Details of GPU MMD library

图 8 GPU MMD 库的详细信息

4.4 GPU 应用程序库

GPU 节点采用 OpenCL 进行核函数的编程，其编程方式与其他支持 OpenCL 编程的计算设备没有区别，这种一致性消除了移植和开发应用程序代码的障碍. 不同的是，Vortex 作为在 RISC-V 的 ISA（instruction set architecture）上实现的 GPU，需要对相应的编译器工具链和应用程序库提供针对性的支持. GPU 内部的软硬件设计工作并非本文所关注的点，因此我们没有修改 GPU 的微架构，遵循了 Vortex 团队提供的库和工具链. 生成 GPU 二进制文件的整个过程如图 9 所示. 终端用户编写的 OpenCL 核函数通过 PoCL 编译器编译生成 LLVM IR. 为了实现任务分配和线程调度的自动化，GPU 核函数运行时库也将被编译，并与核函数生成的 LLVM IR 链接，生成最终的 GPU 二进制文件.

5 系统评估

Direct xPU 系统中的 2 种计算引擎 GPU 和 CPU 都具有专属的网络通信设备，因此传统的通过 PCIe

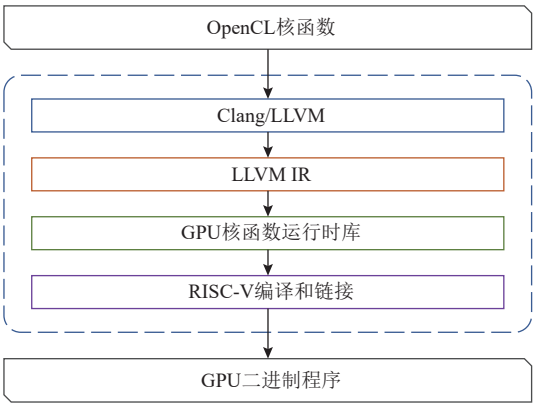


Fig. 9 Generation process of GPU binary program

图 9 GPU 二进制程序的生成过程

连接 GPU 和 CPU 的架构不再合理. 为了在空间上构建一个紧凑的计算资源池，并使其物理形态适应大规模分布式计算中心的需求，我们针对 Direct xPU 开发了一种 xPU Box，如图 10 所示. xPU Box 配备有冷却模块和电源模块，并提供了一定数量的仅供电的 PCIe 插槽，以实现与现有设备物理接口的最大兼容性. 箱内的 GPU 节点通过光纤连接到光纤交换机. 除了图 10 中展示的 16 个 PCIe 插槽的 xPU Box，我们还设计了 4 个 PCIe 插槽的版本，以满足不同规模的部署需求.

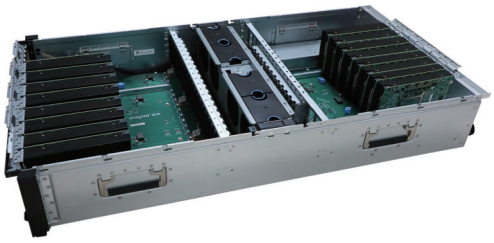


Fig. 10 xPU Box equipped with 16 PCIe slots

图 10 配备了 16 个 PCIe 插槽的 xPU Box

在本次评估中，我们使用了 1 台 PC 作为主机节点，使用 2 个基于 FPGA 的 Direct xPU 原型机作为 GPU 节点. 表 3 给出测试环境的主要软硬件设施的详细信息. 主机节点是一台配备了光纤网卡的 PC，通过 10 Gbps 光纤连接至交换机，2 个 GPU 节点也通过 10 Gbps 光纤连接至交换机. 交换机来自锐捷，其支持最多 48 个设备以 10 Gbps 互联通信. 由于 FPGA 开发板所配备的 MAC 的性能限制，以及开发更高性能的 iRDMA 的硬件资源限制，目前所实现的系统网络性能距离主流的高性能 RDMA 网卡存在一定的差距. 然而，由于 xPU 和 iRDMA 在硬件电路设计上是解耦的，这使得我们可以很容易地在后续迭代升级至更高带宽的版本，这是我们的后续计划. Vortex 支持配置 1~32 个计算核心，每个计算核心包含了 4 个物理线程. 由于 FPGA 芯片的资源限制，本原型中使用了 32 Core 的配置，在 200 MHz 的 FPGA 时钟下，通过执行简单的矩阵乘计算，其实际性能约为 3.68 GFlops.

Table 3 Specification of Main Systematic Components

表 3 主要系统组件的规格

组件	规格
CPU	Intel Xeon E5-2 640 V4
CPU 内存	DDR4，2 133 MHz，256 GB
主机网卡	Intel x520-sr2，10 Gbps
主机操作系统	CentOS 7.8
FPGA 板卡	Inspur F10A（Intel Arria 10）
FPGA 内存	DDR4，2 133 MHz，16 GB
交换机	锐捷 S6220
FPGA 编译工具	Quartus 19.1
Vortex GPU 版本	0.2.3，32 Core

在进行稳定网络环境下的带宽和延迟测试之前，有必要对基于选择性重传的 iRDMA 的设备在不稳定环境中的网络带宽性能进行简单测试. 如 3.3 节中所述，iRDMA 通过更复杂的状态管理，实现了网络性能开销相对更低的选择性重传机制. 商用 GPU 节点间通信工作[2-9] 的通信性能取决于节点内所配备的与 CPU 共享的网络设备的性能，基于 FPGA 的领域专用加速器方的节点间通信工作[10-14] 则未讨论不稳定网络下的重传机制所带来的负面影响. 因此，我们使用相同的 2 个 x86 平台主机，通过模拟不同的网络丢包率，在数据量大小为 1 MB 的条件下，测试了基于 iRDMA 网卡的主机间通信和基于 RoCEv2 的标准网卡——Mellanox ConnectX-4 Lx 10 GB 的主机间通信，在丢包率为 0%，1%，10% 这 3 种情况下网络读写带宽性能的变化情况. 结果如图 11 所示，在丢包率达到 1% 时，基于 Go-back-N 的 Mellanox 网卡已处于几乎不可用的状态，而基于选择性重传的 iRDMA 网络具有更好的可靠性和稳定性.

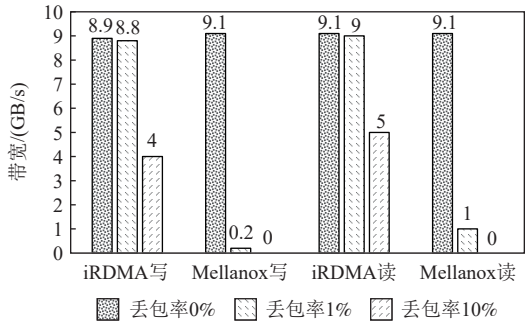


Fig. 11 Bandwidth tests with different network cards at three packet loss rate

图 11 不同网卡在 3 种丢包率下的带宽测试

5.1 稳定网络环境下的带宽和延迟

图 12 展示了发送方视角下 GPU 节点之间完成 1 次通信测试过程所需要的时间和步骤. 在 1 次回环测试中, 发送方如果执行的是写操作, 则发送方发送数据并从接收方接收确认信号 ACK; 发送方如果执行的是读操作, 则发送读命令并接收接收方返回的数据. 延迟主要由 2 阶段组成: 一是主机节点向 GPU 节点发送命令, 以启动 GPU 之间的直接通信, 对应于图 12 中的 t_2–t_0; 二是发送方 GPU 节点向远端的接收方 GPU 节点发送数据/读命令进行写/读操作, 并接收远端的 GPU 节点返回的 ACK/数据, 对应于 t_3-t_2' 部分. $t_2'-t_2$ 段表示 GPU 节点根据命令信息触发网卡的读写操作的时间, 与其他跨芯片传递信息的延迟相比, 由于芯片内模块之间的紧耦合性, 这一阶段通常只需要数十纳秒, 可以忽略.

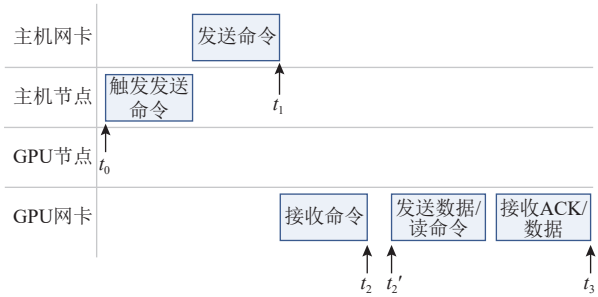


Fig. 12 The complete process of communication between GPU nodes from the perspective of sender

图 12 发送方视角下 GPU 节点间通信的完整过程

图 13 展示了测量 t_3-t_2' 延迟部分的过程. 由于所有节点之间都是使用光纤通过交换机互联, 因此交换机的延迟开销对这一部分的延迟也有一定的影响. 测试结果如图 14 所示. 基于 200 MHz 的 FPGA 时钟频率, 我们通过统计该操作所消耗的时钟数来测量数据包的读/写延迟. 当数据包大小为 4 B 时, 写操作回环时延为 2.65 μs, 读操作回环时延为 2.72 μs, 两者可以认为是相等的.

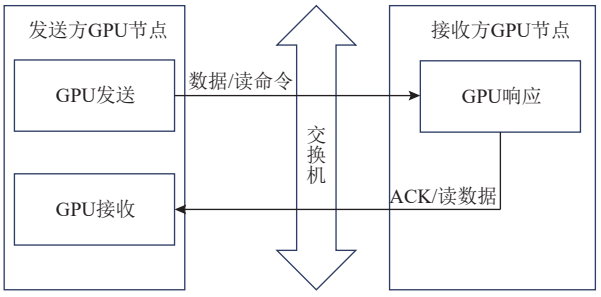
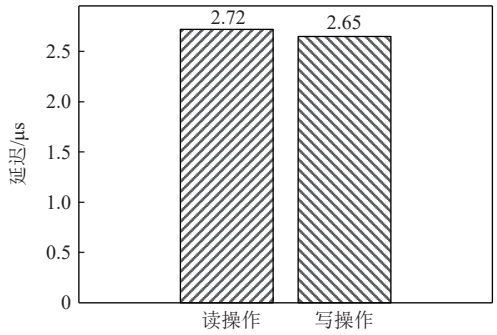


Fig. 13 Latency testing setup between GPU nodes on t_3-t_2'

图 13 GPU 节点之间的 t_3-t_2' 延迟测试设置



GPU节点间读/写操作回环测试

Fig. 14 Latency testing results between GPU nodes communication on t_3-t_2'

图 14 GPU 节点间通信在 t_3-t_2' 的延迟测试结果

据我们所知, 目前广泛部署的商用 GPU 之间的 RDMA 通信是通过与 CPU 共享的网卡实现的, 其延迟包括 CPU 引入的一系列软件栈开销, 并且难以区分或精确测量 GPU 之间通信的延迟组成, 因此, 我们给出了一些基于 FPGA 实现的, 且具备专属通信模块的分布式异构加速器的延迟比较, 如表 4 所示. 值得注意的是, CoE 和 CIRCUS 的延迟将会随着所执行的计算任务的复杂度的增加而增加, 因为更复杂的任务将占用更多的 FPGA 资源, 导致 FPGA 布局布线困难以及时钟频率降低. 然而, 对于基于 GPU 的 Direct xPU 系统, 执行不同的任务仅仅是执行不同的指令, 不会增加硬件复杂度.

Table 4 Comparison of Inter-Node Communicating Latency of Four FPGA-Based Distributed Computing Architectures

表 4 4 种基于 FPGA 的分布式计算架构的节点间通信延迟对比

架构	FPGA 时钟频率/MHz	延迟/μs
CoE	200~250	0.95（4 B 包大小）
CIRCUS	295.8	0.5~1.87（根据包大小和设备的间隔距离变化）
AiS	222.5	5.04（4 B 包大小）
Direct xPU	200	2.69（4 B 包大小）

由于当前实现的 iRDMA 仍依赖于传统的网络协议栈, 导致主机节点依赖于 CPU 进行协议解析和需要操作系统频繁地上下文切换, 因此主机节点与 GPU 节点之间执行通信任务时的延迟 t_2-t_0 相比于 GPU 节点之间的通信延迟大幅度增加. 我们使用 GPU MMD 库的 API 对主机节点和 GPU 节点之间的延迟进行了简单的测试, 即测试主机节点写入 GPU 节点的 CFG 模块中不同数量的寄存器的延迟(每个寄存器为 4 B). 图 15 显示了这部分开销会增加大约

1 ms 的延迟, 并且随着所写入的寄存器数量的增加, 延迟会有轻微的增加, 最终趋于平稳.

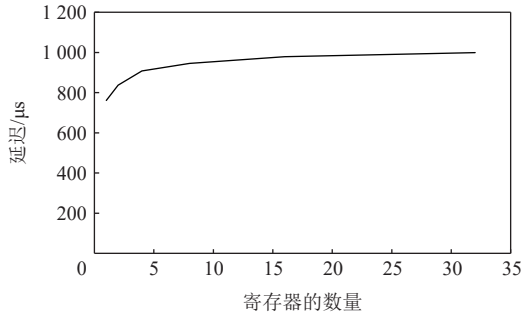


Fig. 15 Latency of different number of registers when the host node writes to the GPU node

图 15 主机节点写入 GPU 节点的不同数量寄存器的延迟

图 16 给出了数据量大小为 1 Mb 和 1 Gb 时, GPU 节点之间的带宽测试结果, 基于 iRDMA 的 Direct xPU 实现了接近 10 Gbps 的理论带宽. 此外, 这种基于网络的通信可以实现几乎相等的读写速度, 避免了 PCIe 链路中读写速度不对等的问题.

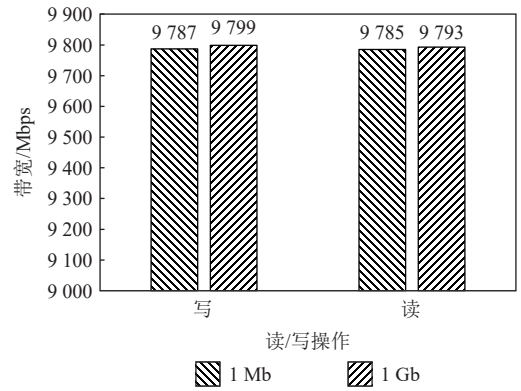


Fig. 16 Bandwidth performance testing result between GPU nodes when the amount of data is 1 Mb and 1 Gb respectively

图 16 数据量大小分别为 1Mb 和 1Gb 时 GPU 节点间的带宽性能测试结果

5.2 原型系统应用演示

我们基于 Direct xPU, 搭建了一个原型演示系统, 如图 17 所示. 图中包含 1 个 4PCIe 插槽的 FPGA Box, 其配备了 2 块 Direct xPU. Direct xPU 通过光纤交换机与 PC 通信. 我们使用了 OpenCL 编程语言将 SqueezeNet 模型进行分割后, 在该分布式计算平台上运行, 即通过摄像头接收图像数据实时识别图像内容.

6 总 结

评估结果表明, 基于 Direct xPU 的原型系统的网络性能基本接近硬件设备的理论上限. 特别是, 我们使用新架构实现了任意 GPU 节点之间的超低延迟、完全独立的通信. 由于原型系统所使用 FPGA 板卡、网络模块等硬件条件限制, 在延迟和带宽性能方面与目前的先进水平还有一定距离.

Fig. 17 Application of a prototype system based on Direct xPU distributed computing platform

图 17 基于 Direct xPU 分布式计算平台的原型系统的应用

可以预见的是, 通过对原型系统中限制其网络性能的硬件设备进行升级来解决其桶效应问题, 各节点之间的通信性能将得到进一步提升. 此外, 得益于 RISC-V 指令集的模块化设计和开源特性, 增加自定义指令使 GPU 节点在指令层面上控制同一芯片内的网络设备成为可能, 这是我们未来的工作计划.

作者贡献声明: 李仁刚设计了论文整体逻辑架构; 王彦伟提出了 Direct xPU 的软硬件架构方案; 郝锐和肖麟阁设计了论文实验方案、完成实验并撰写论文; 杨乐完善论文内容; 杨广文和阚宏伟提出论文撰写指导意见.

参 考 文 献

[1] Fröning H, Nüssle M, Litz H, et al. A case for FPGA based accelerated communication[C]//Proc of the 9th Int Conf on Networks. Piscataway, NJ: IEEE, 2010: 28−33

[2] Shainer G, Ayoub A, Lui P, et al. The development of Mellanox/NVIDIA GPUDirect over InfiniBand—a new model for GPU to GPU communications[J]. Computer Science Research and Development, 2011, 26: 267−273

[3] Ammendola R, Bernaschi M, Biagioni A, et al. GPU peer-to-peer techniques applied to a cluster interconnect[C]//Proc of 2013 IEEE Int Symp on Parallel & Distributed Processing, Workshops and PhD Forum. Piscataway, NJ: IEEE, 2013: 806−815

[4] Agostini E, Rossetti D, Potluri S. Offloading communication control logic in GPU accelerated applications[C]//Proc of 17th IEEE/ACM Int Symp on Cluster, Cloud and Grid Computing (CCGRID). Piscataway, NJ: IEEE, 2017: 248−257

[5] Agostini E, Rossetti D, Potluri S. GPUDirect Async: Exploring GPU synchronous communication techniques for InfiniBand clusters[J]. Journal of Parallel and Distributed Computing, 2018, 114: 28−45

[6] Oden L, Fröning H. GGAS: Global GPU address spaces for efficient communication in heterogeneous clusters[C]//Proc of 2013 IEEE Int Conf on Cluster Computing (CLUSTER). Piscataway, NJ: IEEE, 2013: 1−8

[7] Oden L, Fröning H. InfiniBand Verbs on GPU: A case study of controlling an InfiniBand network device from the GPU[J]. The International Journal of High Performance Computing Applications, 2017, 31(4): 274−284

[8] Daoud F, Watad A, Silberstein M. GPUrdma: GPU-side library for high performance networking from GPU kernels[C]//Procof the 6th Int Workshop on Runtime and Operating Systems for Supercomputers. New York: ACM, 2016: 1−8

[9] Silberstein M, Kim S, Huh S, et al. GPUnet: Networking abstractions for GPU programs[J]. ACM Transactions on Computer Systems, 2016, 34(3): 1−31

[10] Balle S M, Tetreault M, Dicecco R. Inter-kernel links for direct inter-FPGA communication[DB/OL]. [2022-10-19]. https://cdrdv2-public.intel.com/650535/wp-01305-inter-kernel-links-for-direct-inter-fpga-communication.pdf.

[11] Fujita N, Kobayashi R, Yamaguchi Y, et al. Parallel processing on FPGA combining computation and communication in OpenCL programming[C]/Proc of 2019 IEEE Int Parallel and Distributed Processing Symp Workshops (IPDPSW). Piscataway, NJ: IEEE, 2019: 479−488

[12] Fujita N, Kobayashi R, Yamaguchi Y, et al. Performance evaluation of pipelined communication combined with computation in OpenCL programming on FPGA[C]//Proc of 2020 IEEE Int Parallel and Distributed Processing Symp Workshops (IPDPSW). Piscataway, NJ: IEEE, 2020: 450−459

[13] Kobayashi R, Fujita N, Yamaguchi Y, et al. GPU-FPGA heterogeneous computing with OpenCL-enabled direct memory access[C]//Proc of 2019 IEEE Int Parallel and Distributed Processing Symp Workshops (IPDPSW). Piscataway, NJ: IEEE, 2019: 489−498

[14] Kobayashi R, Fujita N, Yamaguchi Y, et al. OpenCL-enabled GPU-FPGA accelerated computing with inter-FPGA communication[C]// Proc of the Int Conf on High Performance Computing in Asia-Pacific Region Workshops. New York: ACM 2020: 17−20

[15] Burstein I. Nvidia data center processing unit (DPU) architecture [C]//Proc of 2021 IEEE Hot Chips 33 Symp (HCS). Piscataway, NJ: IEEE, 2021: 1−20

[16] Sundar N, Burres B, Li Y, et al. An in-depth look at the Intel IPU E2000[C]//Proc of 2023 IEEE Int Solid-State Circuits Conf (ISSCC). Piscataway, NJ: IEEE, 2023: 162−164

[17] Lant J, Navaridas J, Luján M, et al. Toward FPGA-based HPC: Advancing interconnect technologies[J]. IEEE Micro, 2019, 40(1): 25−34

[18] Mittal R, Shpiner A, Panda A, et al. Revisiting network support for RDMA[C]//Proc of the 2018 Conf of the ACM Special Interest Group on Data Communication. New York: ACM, 2018: 313−326

[19] Wadekar M. Handbook of Fiber Optic Data Communication [M]//Cambridge, MA: Academic Press, 2013: 267−287

[20] Jääskeläinen P, de La Lama C S, Schnetter E, et al. PoCL: A performance-portable OpenCL implementation[J]. International Journal of Parallel Programming, 2015, 43: 752−785

[21] Tine B, Yalamarthy K P, Elsabbagh F, et al. Vortex: Extending the RISC-V ISA for GPGPU and 3D-graphics[C]//Proc of the 54th Annual IEEE/ACM Int Symp on Microarchitecture (MICRO-54). New York: ACM, 2021: 754−766

[22] Elsabbagh F, Tine B, Roshan P, et al. Vortex: OpenCL Compatible RISC-V GPGPU[J]. arXiv preprint, arXiv: 2002.12151, 2020

[23] Intel OPAE. Open programmable acceleration engine (OPAE) C API programming guide [DB/OL]. [2021-11-08] https://cdrdv2.intel.com/v1/dl/getContent/686262?explicitVersion=true&wapkw=opae.

[24] Zeng Gaoxiong, Hu Shuihai, Zhang Junxue, et al. Transport protocols for data center networks: A survey[J]. Journal of Computer Research and Development, 2020, 57(1): 74−84 (in Chinese)
(曾高雄, 胡水海, 张骏雪, 等. 数据中心网络传输协议综述[J]. 计算机研究与发展, 2020, 57(1): 74−84)

Li Rengang, born in 1980. PhD, Senior engineer. Member of CCF. His main research interest includes heterogeneous computing.

李仁刚, 1980 年生. 博士, 正高级工程师. CCF 会员. 主要研究方向为异构计算.

Wang Yanwei, born in 1985. PhD, associate researcher. Member of CCF. His main research interest includes heterogeneous computing.

王彦伟, 1985 年生. 博士, 副研究员. CCF 会员. 主要研究方向为异构计算.

Hao Rui, born in 1983. Master, Senior engineer. Member of CCF. His main research interest includes heterogeneous computing.

郝　锐, 1983 年生. 硕士, 高级工程师. CCF 会员. 主要研究方向为异构计算.

Xiao Linge, born in 1994. PhD. His main research interest includes heterogeneous computing.

肖麟阁, 1994 年生. 博士. 主要研究方向为异构计算.

Yang Le, born in 1990. Master. Her main research interest includes heterogeneous computing.

杨　乐, 1990 年生. 硕士. 主要研究方向为异构计算.

Yang Guangwen, born in 1963. PhD, professor, PhD supervisor. Member of CCF. His main research interest includes high performance computing.

杨广文, 1963 年生. 博士，教授，博士生导师. CCF 会员. 主要研究方向为高性能计算.

Kan Hongwei, born in 1975. Master, professor-Senior engineer. Senior member of CCF. His main research interest includes heterogeneous computing.

阚宏伟, 1975 年生. 硕士，教授-正高级工程师. CCF 高级会员. 主要研究方向为异构计算.