计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2022,31(11):320−329 [doi: 10.15888/j.cnki.csa.008809]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

RoCE 协议下基于在网计算的 MPI 通信优化①

李嘉群[1], 蔡文杰[1], 沈　瑜[2], 齐法制[3], 曾　珊[3], 李　京[1,2]

[1](中国科学技术大学 计算机科学与技术学院, 合肥 230027)
[2](中国科学技术大学 超级计算中心, 合肥 230027)
[3](中国科学院 高能物理研究所 计算中心, 北京 100049)
通信作者: 李　京, E-mail: lj@ustc.edu.cn

摘　要: 高性能计算中, 通信上的巨大开销已成为其算力提升的主要瓶颈之一, 通信性能的优化一直是一个重要挑战. 针对通信优化任务, 提出一种基于在网计算技术降低通信开销的方法. 该方法在基于以太网的超算环境下, 利用 RoCEv2 协议、可编程交换机以及 OpenMPI, 实现将归约计算卸载到可编程交换机, 支持 Node 和 Socket 两种通信模式. 在真实超算环境下开展了集合通信基准测试和 OpenFOAM 应用测试实验, 结果表明, 当服务器节点数达到一定规模时, 该方法在 Node 和 Socket 两种模式下相较于传统的主机通信, 均呈现出较好的性能提升, 其中集合通信基准测试有 10%–30% 左右性能提升, 在应用级测试中应用整体性能有 1%–5% 左右提升.

关键词: 在网计算; RoCE 协议; MPI; 通信优化; 高性能计算

引用格式: 李嘉群,蔡文杰,沈瑜,齐法制,曾珊,李京.RoCE 协议下基于在网计算的 MPI 通信优化.计算机系统应用,2022,31(11):320–329. http://www.c-s-a.org.cn/1003-3254/8809.html

MPI Communication Optimization Based on In-network Computing under RoCE Protocol

LI Jia-Qun[1], CAI Wen-Jie[1], SHEN Yu[2], QI Fa-Zhi[3], ZENG Shan[3], LI Jing[1,2]

[1](School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China)
[2](Supercomputing Center, University of Science and Technology of China, Hefei 230027, China)
[3](Computing Center, Institute of High Energy Physics, Chinese Academy of Sciences, Beijing 100049, China)

Abstract: In high-performance computing, the huge communication overhead has become one of the main bottlenecks in the improvement of its computing power, and the optimization of communication performance has always been an important challenge. For the communication optimization task, this study proposes a method based on in-network computing technology to reduce the communication overhead. In the Ethernet-based supercomputing environment, this method utilizes the RoCEv2 protocol, programmable switches, and OpenMPI to offload reduction computation to programmable switches, and it supports the two communication modes of Node and Socket. The collective communication benchmark test and the OpenFOAM application test are carried out in a real supercomputing environment. The experimental results indicate that when the number of server nodes reaches a certain scale, compared with the traditional host communication, this method shows better performance improvement in both Node and Socket modes, with the performance in the collective communication benchmark test improved by about 10%–30% and the overall application performance in the application-level test improved by about 1%–5%.

Key words: in-network computing; RoCE protocol; message passing interface (MPI); communication optimization; high performance computing

① 基金项目: 中科院先导专项 (XDA19020102)
收稿时间: 2022-02-28; 修改时间: 2022-03-28; 采用时间: 2022-04-21; csa 在线出版时间: 2022-07-14

1 引言

随着网络带宽增加、延迟降低以及诸如远程直接内存访问 (RDMA) 等技术的出现, 研究人员开始尝试将科学计算任务中的部分计算操作从 CPU 卸载到特定网络通信设备上 (如可编程交换机), 在高效率利用网络设备高转发能力的同时进一步降低通信开销和 CPU 负载. 因此, 如何充分利用网络设备的计算能力来实现通信优化, 降低通信成本就成为当前高性能计算领域的研究热点之一.

MPI (message passing interface)[1] 是并行科学应用程序设计上的一种常用并行编程模型, 也是高性能计算软件中常用的并行计算应用程序接口. MPI 标准中定义的集合通信操作为组通信操作实现了非常便捷的抽象, 被广泛应用于高性能计算上各种科学领域应用程序中. 这些集合类通信由于涉及全局通信, 往往对应用程序并行效率产生巨大影响. 尽管已有大量研究基于算法优化来提升集合类通信效率[2,3], 如图 1 所示为集合通信 Allreduce 的一种实现算法, 需要节点间两两多轮次互发消息完成通信, 基于算法优化后的方法依然需要在网络中进行多次通信才能完成整体聚合操作, 因此仍无法避免网络拥塞的发生, 并且这些方法优化后的集合类通信延迟相较于点对点通信仍高出一个数量级以上[4]. 在网计算主要通过利用 FPGA、SmartNIC 等网络硬件设备, 使数据在网络上传输的同时在网络设备上进行计算, 从而将集合通信的部分计算卸载到网络通信设备上, 减少通信中路由跳数和网络中传输的数据量, 降低通信总时延.

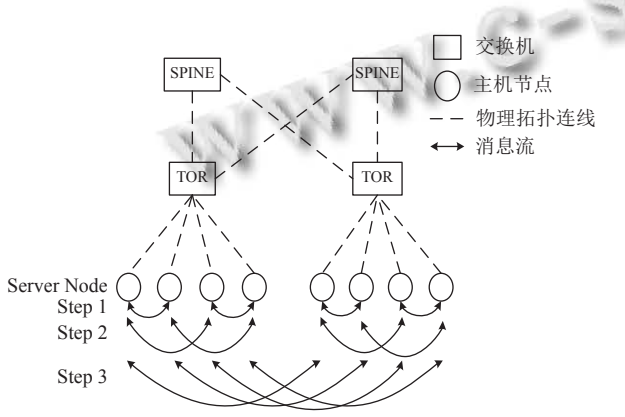


图 1 MPI 集合通信

本文工作主要针对使用 RoCE 协议的超算系统的 MPI 集合通信进行优化. 在开展优化工作前, 调研了大型超算系统 MPI 调用情况[5,6], 同时参考课题组以往对超算应用程序 MPI 调用情况的分析工作[7], 有如下发现:

(1) 由文献 [6,7] 可知, 许多超算应用程序, 如 OpenFOAM、VASP 等, 在 MPI 通信上花费了一半以上的时间;

(2) 与点对点通信相比, MPI 集合通信 (MPI_Allreduce、MPI_Alltoall 等) 花费时间更多, 尤其是 MPI_Allreduce;

(3) 短消息, 即短报文 (≤256 字节) 在超算应用程序 MPI 通信数据包中占比最大.

针对以上集合通信特征, 本文面向 MPI 集合通信优化提出一种基于在网计算的 MPI 通信优化方法, 在 RoCE 网络协议下, 实现基于在网计算的 MPI 集合通信优化. 本文主要工作包括以下 3 个:

(1) 基于 RoCE 协议设计扩充支持在网计算的通信协议, 将 MPI_Allreduce 集合通信的计算操作卸载到可编程交换机进行;

(2) 基于 OpenMPI-4.0.5 扩展实现 MPI_Allreduce 集合通信的在网计算 Node 与 Socket 模式功能;

(3) 在中科大超级计算平台对在网计算功能进行基准测试验证和对 OpenFOAM 进行应用级性能验证.

2 相关研究

针对 MPI 集合通信优化的研究, 研究人员们已开展大量工作来提升通信性能, 并取得了不错的成果. MPI 集合通信优化的研究工作主要可分为 3 类.

(1) 通过改进算法来优化 MPI 集合通信

Vadhiyar[3] 提出使用顺序、链、二叉树和 Rabenseifner 算法实现 Reduce、Gather 和 Bcast 操作; Hoefler 等人[8] 研究无阻塞 Allreduce 操作的几种实现, 展示使用大型通信器和大型消息时的性能提升; 百度为优化深度学习提出 Ring Allreduce; NCCL 贴合硬件实现 Allreduce[9], 其实现算法和 Ring Allreduce 算法相似; 腾讯提出分层的 Ring Allreduce[10], 对节点进行分组, 通过组内 Reduce, 组间 Allreduce, 然后组内 Bcast 的步骤, 完成 Allreduce 操作. 索尼提出 2D-Torus Allreduce 算法[11], 主要思想也是分层, 通过组内 Scatter-Reduce, 组间 Allreduce, 组内 Allgather 完成 Allreduce 操作. Google 提出 2D-Mesh Allreduce 算法[12], 通过两步水平和垂直的环来完成 Allreduce 操作. 该类优化算法虽然可以降低集合通信的时间复杂度, 但通信时间仍随服

务器数目增加而递增, 当服务器数量较多时, 需在网络中进行多次通信, 导致网络中传输的数据量较多.

(2) 针对特定网络拓扑来优化集合通信

该类方法比较典型的是 Barnett 等人[13] 和 Liu 等人[14] 的研究, 他们分别针对网格拓扑和超立方拓扑进行优化. 该类优化主要针对特定网络拓扑的特点, 改进算法以优化 MPI 通信, 但该方法通用性较差, 不适用于普通网络拓扑结构的集合通信优化.

(3) 基于硬件优化集合通信

通常, 大多数 MPI 是通过节点的 CPU 进行集合通信数据的计算, 而网络仅用于数据传输. 基于硬件来优化集合通信相对较少, Quadrics[15] 在网络设备硬件中实现对 Bcast 和 Barrier 的支持. IBM 的 Blue Gene 超级计算机实现了对 Barrier 和 Reduce 的网络级硬件支持, Blue Gene/L[16] 对于 Alltoall 集合通信提升高达两倍的吞吐量性能. IBM 的 PERCS 系统[17] 完全将集合归约操作卸载给了硬件. Mai 等人提出了 NetAgg 平台[18], 它使用网络内盒进行分区聚合操作, 以提供高效的网络链路利用. Cray 的 Aries 网络[19] 在 HCA 中实现了 64 字节 Reduce 支持, 支持基数为 32 的归约树. NVIDIA Mellanox 从 EDR InfiniBand 交换机开始引入的 SHARP 技术[20–22], 旨在将集合通信的计算卸载到网络上, 从而提高计算效率, 降低通信时延.该类从硬件角度出发的优化方法, 其优化效果明显, 其中一些工作是为高性能计算应用程序定制的互连架构开发的, SHARP 技术基于 InfiniBand[23] 实现在网计算, 其网络设备采购成本过高, 且作为商业产品, 代码闭源, 设计和实现的细节未公开, 难以在其上扩展新功能, 且无法在以太网上使用.

综上所述, 现有 MPI 集合通信优化方法都有其局限性, 本文提出在 RoCE 协议下, 采用传统以太网网络, 通过将计算卸载到可编程交换机上以提升集合通信性能, 实现数据边传输边计算, 降低网络中传输的数据量, 且能够充分利用当前丰富的以太网资源和硬件优化的优势, 实现低成本高性能集合通信优化的目的, 同时也为基于算法与拓扑结构的通信优化提供了更大的性能提升空间.

3 基于在网计算的 MPI 通信优化设计与实现

3.1 在网计算设计简介

本文目标是通过在网计算技术实现 MPI_Allreduce 集合通信的优化, 在通信过程中将部分计算交于可编程交换机执行, 此时交换机不仅可以转发报文, 同时还承担简单的计算工作. 在网计算的基本原理如图 2 所示, 以两层交换机组网为例, Server Node 层的主机节点发送数据时必经过交换机 (节点间无直接通信), 通过算法将某些交换机作为聚合节点参与计算. 如图 2 中一层交换机 (TOR_1 与 TOR_2) 均为聚合节点, 可按照一定规则选取某一个二层交换机 (如 SPINE_1) 为聚合节点, 具有计算功能的交换机接收到数据后, 对需要在网计算的数据进行缓存并进行相应的计算, 计算完成后, 将最终计算结果向下层主机节点传递. 利用可编程交换机参与计算可以减少转发的数据量和主机节点的计算量, 达到节省带宽降低时延的效果, 对提高主机节点的计算能力也有帮助. 基于在网计算的 MPI_Allreduce 通信有如下优点:

(1) 减少网络中通信路由跳数;

(2) 降低总体通信时延;

(3) 减少网络中传输的数据量及主机节点的计算负载.

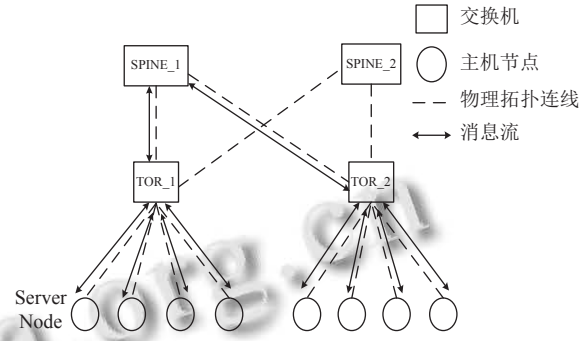


图 2 基于在网计算的 MPI 通信

本文针对 MPI 集合通信优化, 设计并实现了基于在网计算的 MPI 通信优化方案, 主要包括: 通信报文设计、控制面和数据面设计.

通信报文是在网计算功能实现的载体, 通信报文基于 RoCEv2 协议进行扩充, 其处理架构如图 3 所示. 可编程交换机可看成由输入队列、仲裁器、控制器、输出队列、FPGA 几个部分组成. 输入队列和输出队列用于暂存数据包. 仲裁器按照包的优先级从输入队列中获取数据包, 并根据报文标识判断是普通报文还是在网计算报文. 如果是普通报文则直接转发; 如果是在网计算报文则转发给 FPGA, FPGA 对报文进行解析, 对控制报文和数据报文进行相应处理. FPGA 处理完后将结果形成新的报文转发.

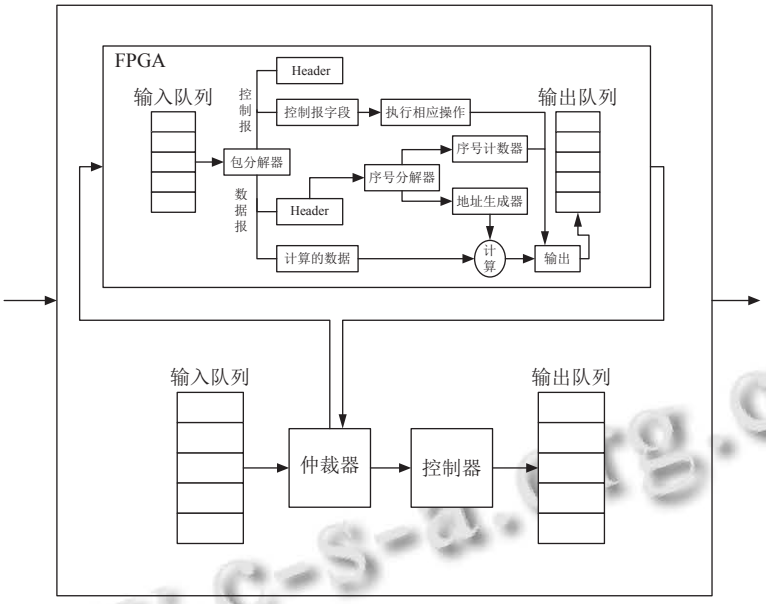


图 3 在网计算协议处理架构

控制面主要完成协商控制, 即通过可编程交换机参与节点间控制报文的通信过程, 在交换机上创建相应路由表, 根据路由表建立聚合计算逻辑树用于在网计算, 同时主机节点获得可编程交换机关于在网计算的相关参数, 如交换机支持的计算操作类型、数据类型等; 数据面主要负责数据报文的发送与计算. 在网计算的整体流程如图 4 所示. 在网计算支持 Node 与 Socket 两种模式, Node 模式仅从节点层面进行考虑, 每个节点选取一个领导者进程用于控制面节点间通信以及数据面节点内进程的聚合和广播. Socket 模式是对 Node 模式的进一步优化, 考虑到一个节点下有多个 CPU 的情况, 每个 CPU 下都选取一个领导者进程用于控制面和数据面节点间并行通信以及数据面同一 CPU 内进程的聚合和广播. Socket 模式可以将同一个 CPU 下的进程划分为同一个通信子域, 降低跨 CPU 通信的开销, 同时提高节点间通信的并行性.

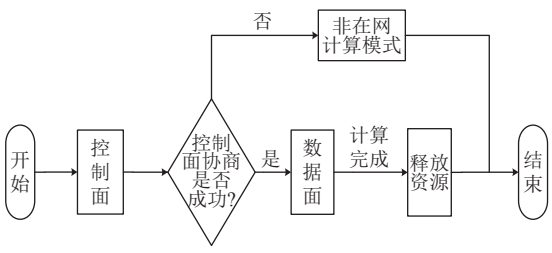


图 4 在网计算流程

3.2 通信报文设计

为使可编程交换机识别控制报文和数据报文, 实现在网计算功能, 需要设计新的通信协议. 通信协议的设计是对当前以太网 RoCE 协议的扩充, RoCE (RDMA over converged Ethernet) 是 RDMA 技术在以太网上的一种实现方法. RoCE 是一种机制, 其提供在无损以太网上实现极低延迟的高效通信方法. RoCE 协议有 RoCEv1 和 RoCEv2 两个版本, 其差别取决于系统所采用的网络适配器或网卡.

本文通信协议基于以太网的 RoCEv2 协议进行设计. 协议设计报文字段如图 5 所示, 基于 RoCEv2 协议增加 MPI 头和 payload 字段, 协议支持 RC 和 UD 两种模式.

UD	ETH (14 B)	VLAN (4 B)	IP (20 B)	UDP (8 B)	BTH (12 B)	DETH (8 B)	UCP (8 B)	UCP TAG (8 B)	MPI (22 B)	payload	ICRC (4 B)	FCS (4 B)
RC	ETH (14 B)	VLAN (4 B)	IP (20 B)	UDP (8 B)	BTH (12 B)	ID (1 B)	UCP TAG (8 B)	PAD DING (15 B)	MPI (22 B)	payload	ICRC (4 B)	FCS (4 B)

图 5 通信报文格式

图 5 中 MPI 头和 payload 主要用于定义一些字段以表示在网计算报文的相关信息, 其中 MPI 头定义在网计算报文的通用字段, payload 在控制报文和数据报文情况下有所不同, 控制报文的 payload 定义了需要协商的参数, 而数据报文的 payload 仅用于保存将要进行计算的真实数据. 图 5 中圆括号内为报文各字段大小.

MPI 头的关键字段及其相应含义如表 1 所示, 控制报文 payload 的关键字段及其含义如表 2 所示.

表 1 MPI 头关键字段及含义

字段	大小 (B)	含义
tag	4	数据、控制报文有效. 在网计算报文的特殊标识
src_rank	4	数据、控制报文有效. 表示源进程Rank
comm_id	2	数据、控制报文有效. 控制报文为特定值, 数据报文为在网计算群组ID
op_code	1	数据报文有效, 表示操作符类型, 例如MPI_MAX, MPI_MIN
oper_and_type	1	数据报文有效, 表示操作的数据类型, 例如INT16/INT32等
req_id	1	数据报文有效, 对齐所有服务器传输的消息, req_id相同的, 进行在网计算
pkt_para_num	2	数据报文有效, 表示参数个数, 用于明确payload中携带了几个有效数据

表 2 控制报文 payload 关键字段及含义

字段	大小 (B)	含义
master_num	1	Node/Socket模式为0/1
query_notify_hop	1	标识请求报文和响应报文及消息经过交换机的次数
sup_comm_type	1	支持的在网计算集合操作类型, 当前支持Allreduce
sup_allreduce_op	2	支持的操作函数, 交换机更新
sup_data_type	2	支持的数据类型, 交换机更新
sup_max_data_size	2	最大支持256个字节的有效数据
global_group_size	2	参与运算的服务器总数量
local_group_size	2	同一TOR下服务器的数量
true_comm_id	4	不同在网计算通讯域的群组ID, 随机值
ava_grp_num	4	负载信息, SPINE填写自身还能支持多少在网群组

表 2 中, 控制报文 payload 字段除了上述关键字段以外, 还有几个非核心字段未详细列出, 包括: sup_mpi_type 表示支持的 MPI 版本; fail_cause 表示协商是否成功; dst_rank 表示目的地的 Rank 号 (Rank 表示进程号, 由 MPI 分配, 用于区分一个通信域内的进程); tor1_ip 表示经过第一个 TOR 交换机的 IP 地址; spine_ip 表示经过 SPINE 交换机的 IP 地址; tor2_ip 表示经过第 2 个 TOR 交换机的 IP 地址; job_id 表示调度器传入的 MPI 作业的编号, 以区分不同的任务; c_id 表示子通信域的 ID; world_rank 表示在整个通信域中的 Rank 号.

3.3 控制面设计

在网计算实现流程中, 控制面的作用主要是让主机节点与网络中的可编程交换机进行交互, 交互过程中主机节点获得交换机上关于在网计算的相关配置参数等信息, 而交换机则根据节点消息, 掌握系统中网络节点分布与节点状态等信息, 建立通信路由表, 生成聚合计算逻辑树, 为后续数据面流程提供通信基础信息.

在网计算控制面和数据面通信流程中存在多个进程间交互, 为便于描述, 对本节和下节中涉及到的相关进程, 统一规定如表 3 所示.

表 3 Node 和 Socket 模式下进程描述

进程	模式	含义
SRR	Node	每一主机节点下Rank号最小的进程
	Socket	每一主机节点各CPU下Rank号最小的进程
SR	Node/Socket	除*SRR*进程外, 参与在网计算主机节点下的其它进程
MRR	Node	*SRR*进程中Rank号最小的一个进程
	Socket	*SRR*进程中Rank号最小的*N*个进程, *N*为一个主机节点下CPU数目
NMRR	Node/Socket	在*SRR*进程中, 除*MRR*进程外的其余进程

本节和下一节详细介绍 Node 模式下控制面和数据面的设计, Socket 模式下控制面和数据面与 Node 模式原理大体相似, 相关差异部分在第 3.5 节中单独介绍. Node 模式下在网计算需满足节点进程 Rank 全局均匀且连续的约束 (该约束可通过 OpenMPI 实现), 即:

(1) 在参与在网计算的主机节点中, 同一节点下运行的进程数相同且进程 Rank 号连续;

(2) 不同主机节点上进程 Rank 号连续;

(3) 同一 TOR 交换机下不同主机节点进程 Rank 号连续.

对图 6 中在网计算控制面核心流程进行详细介绍.

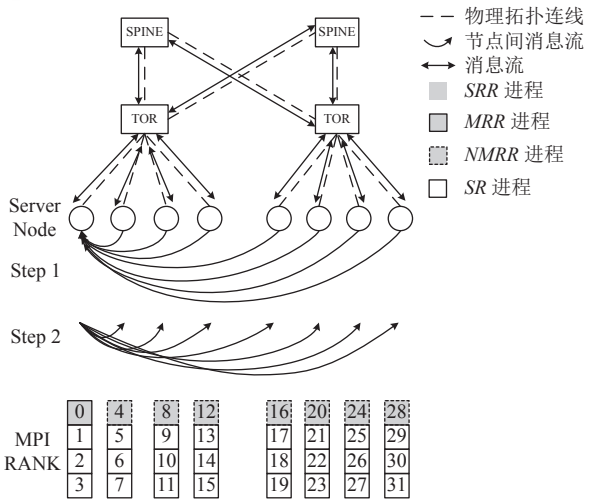


图 6 在网计算控制面流程

(0) Step 0, 控制面初始化阶段. 该阶段会初始化参与在网计算主机节点下的进程, 构建通信域, 将进程划分为 *SRR* (对应前文领导者进程, 包括 *MRR* 和 *NMRR* 两种) 和 *SR* 进程, 然后 *MRR* 进程会随机生成通信群组的唯一标识 ID, 用于标识在网计算群组, 并经由 MPI 底层点对点通信发送至其他所有进程;

(1) Step 1, query 阶段. 所有 *NMRR* 进程均经交换机发送在网计算请求报文至 *MRR* 进程. 请求报文途经 TOR 和 SPINE 交换机, 交换机会对请求报文进行解析处理, 填入其支持的计算类型、数据类型等信息后形成新报文转发. *MRR* 收集请求报文并进行 SPINE 节点选择 (选择通信负载最小的 SPINE), 保存 SPINE 节点信息用于 notify 阶段, 此时 query 阶段完成;

(2) Step 2, notify 阶段. 当 *MRR* 进程收集完所有 *NMRR* 进程的请求报文后, 会根据请求报文判定 SPINE 交换机的选取, 及确定所有交换机共同支持的在网计算数据类型、计算类型等信息, 写入在网计算响应报文并经交换机向 *NMRR* 进程发送. 当 *NMRR* 进程收到响应报文后, 对报文信息进行保存, 并根据响应报文信息确定是否可以进行在网计算. 若节点所连交换机不具有计算能力, 则转入传统 MPI 通信模式; 若交换机具有计算能力, 则 *MRR* 及 *NMRR* 进行其各自节点内进程的广播, 将获取的在网计算参数通过广播方式发送给同节点下其他 *SR* 进程. 本文中广播阶段通信采用 OpenMPI 的二叉树广播方式实现. 此时, notify 阶段完成;

上述所有阶段完成后, 控制面流程结束, 可编程交换机路由表建立完毕, 聚合计算逻辑树生成, 为后续数据面流程开展建立了通信基础. 此时, 可转入数据面流程进行在网计算数据报文的发送与计算工作.

需要注意的是, 控制面还需在数据面计算流程全部结束后, 对交换机上路由表及申请的资源进行释放. 该资源释放操作由 *MRR* 进程经由交换机向 *NMRR* 进程发送释放资源的 kill 报文, 报文在传输到交换机处时, 交换机解析报文识别到 kill 信息后, 自动对其资源进行释放.

3.4 数据面设计

在网计算数据面流程主要进行实际业务数据发送及相关计算工作. 当控制面完成后, 可正式通过数据面发送数据, 数据将在可编程交换机处进行聚合计算, 计算结果再发送至主机节点.

对图 7 中在网计算数据面流程进行详细介绍.

(0) 在节点间发送数据报文之前, 每个主机节点的 *SRR* 进程会先进行一次 Reduce 操作, 对同一个主机节点下其他进程的数据进行聚合计算;

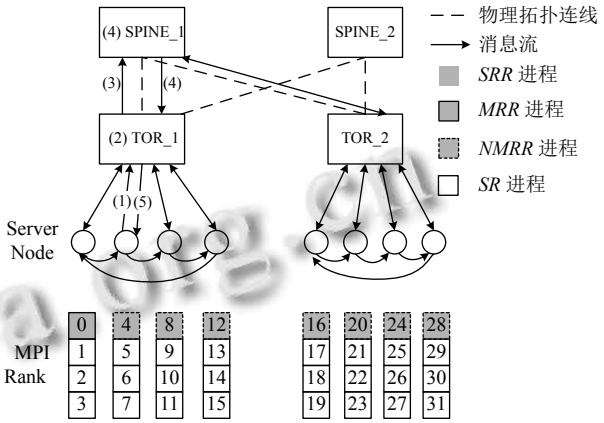


图 7 在网计算数据面流程

(1) 主机节点间经由交换机发送数据报文, 同一 TOR 下的 *SRR* 进程, 按照 Rank 号从小到大的顺序首尾相连闭环发送数据报文, 报文发送过程中会经过 TOR, 主机节点间无直接通信;

(2) TOR 缓存报文, 并进行 Allreduce 计算操作;

(3) TOR 判断与其直连节点的同一轮报文收集并计算完成后, 向 SPINE 节点发送初步计算完的数据报文;

(4) SPINE 缓存报文, 进行 Allreduce 计算操作, 判断与其直连 TOR 的同一轮报文收集并计算完成后, 发送最终的计算结果给 TOR;

(5) TOR 将收到的最终计算结果, 填入缓存的原始报文中, 发送给主机节点 *SRR* 进程. 最后 *SRR* 进程将结果广播至位于同一主机节点下其它进程, 此时一轮 Allreduce 完成.

3.5 Socket 模式设计

上述描述为 Node 模式下控制面和数据面, 为进一步提升通信性能, 本文提出在网计算 Socket 模式, 但其使用的通信环境需满足以下约束条件:

(1) 参与在网计算的每个主机节点有两个及以上物理 CPU, 且每个主机节点的 CPU 数目相同;

(2) 每个 CPU 下运行相同数目的进程且进程 Rank 连续, 每个主机节点下多个 CPU 运行的进程 Rank 连续;

(3) 不同主机节点下进程的 Rank 连续;

(4) 同一 TOR 下不同主机节点进程的 Rank 连续.

Socket 模式下控制面和数据面通信与 Node 模式有所不同, Socket 模式下将 *SRR* 进程按照节点下 CPU 的顺序分成多个组, 按组进行控制面和数据面的通信.

下面将通过 Socket 模式实现流程对上述分组的通信方法进行详细介绍. 设一个主机节点下的 CPU 数目为 N, 主机节点的总数量为 S, 每个 CPU 下运行 P 个进程. 则 *MRR* 的大小为 $MRR_{\text{size}}=N$, *NMRR* 的大小为 $NMRR_{\text{size}}=S\times N-N$. 其中 *MRR* 和 *NMRR* 按照 Rank 从小到大排序 (此处 *MRR* 和 *NMRR* 为进程集合), 则 *MRR* 和 *NMRR* 可以如下表示:

$$MRR=\{i\times P|i\in\{0,1,\cdots,N-1\}\} \tag{1}$$

$$NMRR=\{i\times P|i\in\{N,N+1,\cdots,S\times N-1\}\} \tag{2}$$

其中, $i\times P$ 表示各进程 Rank 号.

Socket 模式下控制面 query 阶段和 notify 阶段 *MRR* 进程和 *NMRR* 进程报文交互分别按照下述对应关系进行:

$$NMRR_i\rightarrow MRR_j\left\{\begin{array}{l}j=(i-1)\%N+1\\ i\in\{1,2,\cdots,S\times N-N\}\end{array}\right\} \tag{3}$$

$$MRR_i\rightarrow NMRR_j\left\{\begin{array}{l}j=k\times N+i\\ k\in\{0,1,\cdots,S-2\},i\in\{1,2,\cdots,N\}\end{array}\right\} \tag{4}$$

query 阶段时, *NMRR* 中进程按照式 (3) 中对应关系发送请求报文给 *MRR* 中进程, 式中 *NMRR* 的第 i 个进程和 *MRR* 中第 j 个进程均属于第 j 组, *NMRR* 进程将请求报文发送给同组的 *MRR* 进程. 由于 Socket 模式中 $N\geqslant 2$, 即存在多个 *MRR* 进程, 因而 *MRR* 进程间还需将保存的通信信息进行同步后再进行 notify 阶段. notify 阶段, *MRR* 进程按照式 (4) 中对应关系发送响应报文给 *NMRR* 进程, 式中 *MRR* 中第 i 个进程及 *NMRR* 的第 j 个进程属于第 i 组, *MRR* 进程发送响应报文给同一组内的所有 *NMRR* 进程.

Socket 模式下进行数据面交互时, 通信方式与 Node 模式类似, 差别在于 Socket 模式同一 TOR 下呈多个组闭环发送形式. 设同一 TOR 下服务器数量为 S', 同一 TOR 下的 *SRR* 进程集合为 SRR', $SRR'_{\text{size}}=S'\times N$, 其中 SRR' 按照进程 Rank 从小到大排序. 数据面的报文发送按照式 (5)、式 (6) 进行.

$$SRR'_i\rightarrow SRR'_j\left\{\begin{array}{l}j=i+N\\ i\in\{1,2,3,\cdots,(S'-1)\times N\}\end{array}\right\} \tag{5}$$

$$SRR'_i\rightarrow SRR'_j\left\{\begin{array}{l}j=(i-1)\%N+1\\ i\in\{(S'-1)\times N+1,\cdots,S'\times N\}\end{array}\right\} \tag{6}$$

数据面交互时, 首先 SRR' 进程聚合同一 CPU 下其他进程的数据, 然后 SRR' 中进程按照式 (5)、式 (6) 发送数据报文, 式中 SRR' 进程的第 i 个进程与第 j 个进程属于同一组, 进程按照同组内闭环的形式发送数据报文. 当数据报经过交换机时, 交换机对数据进行计算, 并将最终计算结果再发给 SRR' 中进程, 然后 SRR' 进程将计算结果广播给同一 CPU 下的其他进程.

图 8 以二层交换机 Socket 模式为例, 对上述流程中各参数进行介绍. 其中 N=2, S=9, P=4, *MRR*={0, 4}, *NMRR*={8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 56, 60, 64, 68}, $SRR=MRR\cup NMRR$, *SR* 为图 8 所有 72 个进程中除 *SRR* 以外的其他进程. 以第一个 TOR 为例, $S'=3$, 集合 $SRR'=\{0,4,8,12,16,20\}$, 其他 TOR 下的情况以此类推. 根据图 8 解释 Socket 模式的通信流程, 在控制面中 *NMRR* 进程分为{8, 16, 24, 32, 40, 48, 56, 64}和{12, 20, 28, 36, 44, 52, 60, 68}两组, *MRR* 进程分为{0}和{4}两组, 第 1 组 *NMRR* 与 *MRR* 进程 0 交互, 第 2 组 *NMRR* 与 *MRR* 进程 4 交互. 数据面中第一个 TOR 下 *SRR* 进程分为{0, 8, 16}和{4, 12, 20}两组分别呈闭环发送数据.

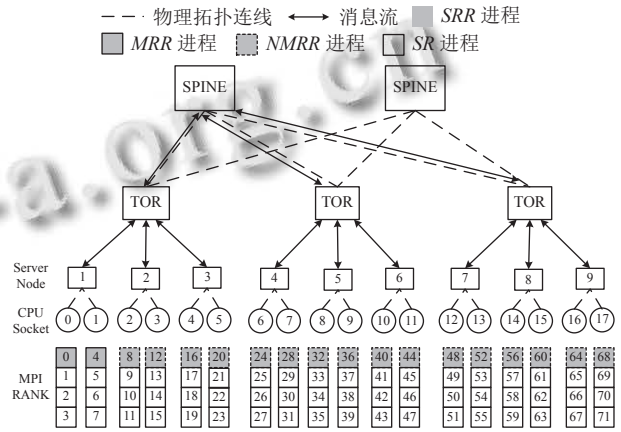


图 8 在网计算 Socket 模式

4 实验及结果分析

为验证在网计算优化效果, 本文主要针对在网计算 MPI 集合通信进行基准测试和相应的应用级测试. 主要测试在不同主机节点数量、不同消息大小、以及主机节点运行不同进程数目情况下的时延, 并与非在网计算实验的结果进行比较, 进而分析性能提升效果.

4.1 实验环境

本文实验于中科大超级计算中心的瀚海 20 系统上进行, 实验主要采用国产 ARM CPU 计算节点, 节点的型号是华为 Taishan 2280V2, 节点的主要参数为 2×海思 Hi1620 CPU (48 核, 2.6 GHz), 256 GB DDR4 2 666 MHz 内存, 300 GB SAS 硬盘, ARM 节点之间通过 25 Gb/s 的以太网互联. 交换机型号为华为 CE8850-64CQ-EI, 提供高密度 100GE/40GE/25GE/10GE 端口, 软件平台基于 VRP8 操作系统. FPGA 型号为 Intel A10 系列, 实验采用的 MPI 版本为 OpenMPI-4.0.5.

4.2 集合通信基准测试

本文基准测试实验使用 OSU-Micro-Benchmarks 基准套件进行 Allreduce 的基准测试. 表 4–表 6 分别为在迭代 10 000 次情况时, 测试在 4 个主机节点、9 个主机节点以及 16 个主机节点不同进程数 (*PPN*)、不同消息大小的集合通信基准测试实验结果. 实验对比了在网计算模式 (包括 Node 模式和 Socket 模式) 及非在网计算模式下 osu_allreduce 的平均延迟情况. 其中消息大小 (数据面数据报文大小) 最大为 256 字节, 时延以微秒 (μs) 为单位.

表 4 4 节点下非在网计算和在网计算对比实验

4节点消息大小 (B)	非在网计算 (μs)			在网计算Node模式 (μs)			在网计算Socket模式 (μs)		
	PPN=8	*PPN*=32	*PPN*=64	*PPN*=8	*PPN*=32	*PPN*=64	*PPN*=8	*PPN*=32	*PPN*=64
4	6.89	10.45	16.44	8.45	11.79	14.72	6.4	9.44	14.07
8	6.77	9.36	14.48	8.38	10.66	13.99	6.36	9.07	13
16	6.81	9.43	14.63	8.43	10.68	13.97	6.53	8.98	12.35
32	7	9.72	15.87	9.28	11.47	15.3	6.8	9.55	13.15
64	7.52	11.33	18.35	9.66	11.75	15.79	7.05	9.76	14.52
128	8.62	12.4	19.28	10.25	12.82	16.44	8.35	10.82	14.62
256	10.73	19.03	28.88	12.27	14.97	19.01	9.75	12.74	16.5

表 5 9 节点下非在网计算和在网计算对比实验

9节点消息大小 (B)	非在网计算 (μs)			在网计算Node模式 (μs)			在网计算Socket模式 (μs)		
	PPN=8	*PPN*=32	*PPN*=64	*PPN*=8	*PPN*=32	*PPN*=64	*PPN*=8	*PPN*=32	*PPN*=64
4	9.9	15.68	20.66	8.63	12.79	14.78	7.16	11.67	14.05
8	9.8	13.53	20.51	8.6	11.05	14.87	7.19	9.91	14.13
16	9.89	13.61	20.86	8.73	11.08	15.19	7.35	10.01	14.05
32	10.95	13.86	21.34	10.43	12.12	15.83	8.51	10.88	15.07
64	11.25	17.43	27.21	9.84	12.36	16.45	8.15	11.27	16.79
128	12.05	18.61	31.35	10.78	13.84	17.63	9.19	12.4	17.31
256	15.61	30.4	43	12.89	16.4	20.21	11.55	14.92	19.33

表 6 16 节点下非在网计算和在网计算对比实验

16节点消息大小 (B)	非在网计算 (μs)			在网计算Node模式 (μs)			在网计算Socket模式 (μs)		
	PPN=8	*PPN*=32	*PPN*=64	*PPN*=8	*PPN*=32	*PPN*=64	*PPN*=8	*PPN*=32	*PPN*=64
4	11.53	18.51	23.98	9.91	15.53	17.15	8.3	14.03	15.96
8	11.29	15.73	24.75	9.8	12.59	17.03	8.18	11.27	15.51
16	11.58	15.75	24.32	9.83	12.45	16.72	8.34	11.37	15.43
32	12.37	16.11	24.72	11.68	13.28	17.6	9.93	12.2	16.3
64	12.99	20.85	33.85	10.84	13.56	18.1	9.32	12.59	16.91
128	13.85	22.76	39.33	12.26	15.25	19.92	11.05	14.27	19.14
256	18.22	36.09	53.73	14.59	18.2	22.35	14.18	17.37	22.34

由表 4 可以看出在 4 个主机节点的情况下, 在网计算 Node 模式相较于非在网计算优化效果不明显, 当 *PPN* 为 64 时 Node 模式有 3%–15% 左右的性能提升. 而 Socket 模式相较于非在网计算有 2%–25% 左右的性能提升.

由表 5 可以看出, 随着节点数增加, 在 9 节点情况下, 在网计算 Node 模式和 Socket 模式相较于非在网计算, 时延明显有所降低, 其中 Node 模式相较于非在网计算大约有 10%–20% 左右的提升, 而 Socket 模式相较于非在网计算有 20%–30% 左右的性能提升, 且随着 *PPN* 和消息大小的增加, 提升越大.

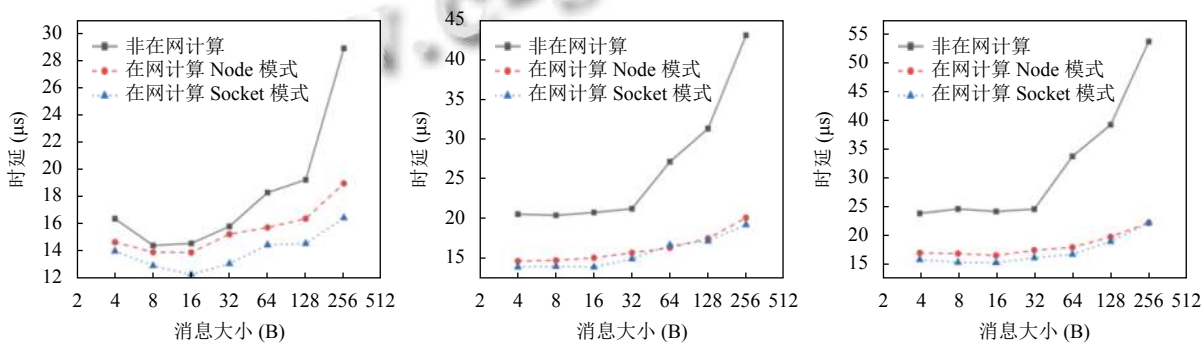
由表 6 可以看出, 16 节点情况下, Node 模式和

Socket 模式相较于非在网计算, 提升更为明显. 16 节点情况下, Node 模式相较于非在网计算, 大约有 15%–30% 的性能提升, Socket 模式相较于非在网计算有 25%–35% 的性能提升, 同样随着消息大小和 *PPN* 的增加, 提升越大.

基准测试实验表明, 本文提出的在 RoCE 协议下基于在网计算的 MPI 通信优化方法, 在当主机节点数目大于等于 9 时, 其优化效果明显, 性能提升显著. 图 9 是 *PPN* 为 64 时不同节点情况下的实验结果图. 横坐标为消息大小, 纵坐标为时延, 可以直观看出在网计算的效果.

4.3 OpenFOAM 应用测试

为了进一步验证本文在网计算方法对 MPI 通信优化提升效果, 进行了相关应用级性能测试. 本文主要针对中科大超算中用户数占比较高的集合通信应用 OpenFOAM 进行测试. 本文实验测试采用 OpenFOAM 自带的流体力学算例 Lid Driven Cavity Flow, 采用 icoFoam 求解器进行求解. 选择该算例是因为, 通过统计分析发现 MPI_Allreduce 通信时间占比较大, 调用次数相对较多, 且消息大小小于 256 字节的短消息占比较大, 满足本文实验要求. 表 7 展示了 OpenFOAM 性能测试结果, 主要测试在 9 节点和 16 节点情况下非在网计算和在网计算模式不同 *PPN* 的运行总时间, 时间单位为 s. 可以看出, 在 9 节点情况下, 在网计算大约有 1%–4% 的性能提升, 在 16 节点情况下, 在网计算大约有 3%–5% 的性能提升. 该性能提升为在网计算模式下应用总的运行时间相较于非在网计算模式下的提升.



(a) 4 节点 (b) 9 节点 (c) 16 节点

图 9 *PPN*=64 不同节点情况下实验结果图

表 7 OpenFOAM 性能测试实验

PPN	9节点 (s)		16节点 (s)	
	非在网计算	在网计算	非在网计算	在网计算
2	203.92	201.03 (1.42%)	184.32	178.64 (3.08%)
4	197.43	192.32 (2.59%)	236.01	227.03 (3.80%)
8	270.56	260.78 (3.61%)	342.36	327.08 (4.46%)
16	287.74	276.01 (4.08%)	334.12	317.07 (5.10%)
32	315.01	301.59 (4.26%)	417.36	393.66 (5.68%)
64	385.4	369.88 (4.03%)	450	428.45 (4.79%)

5 总结与展望

随着智能化、信息化时代的高速发展, 人工智能等新兴技术的发展促使将计算操作卸载到网络设备成为研究热点之一. 本文提出在 RoCE 网络协议下在网计算优化 MPI 通信的设计及其实现, 通过集合通信基准测试及 OpenFOAM 应用测试, 验证本文在网计算通信优化方法的有效性. 实验表明, 本文在网计算方法的 Node 模式和 Socket 模式均有一定通信性能提升.

然而, 在网计算也有其自身局限性. 从实验结果可以发现当节点数量较多时, 在网计算有一定的提升. 但当节点数量较少时, 在网计算优势不明显. 当节点数在 9 以上时, 在网计算相较于主机上进行计算优势较为明显. 但是由于交换机相对于主机的局限性, 当进行复杂计算、计算数据量很大、网络状况较好时, 将计算卸载到交换机上执行可能无法减少总时间, 因此后续可进一步研究实现交换机的自适应卸载. 目前本文仅基于硬件进行通信优化, 后续可以将基于算法和拓扑的优化和在网计算进行结合, 进一步提升优化效果. 此外, 本文设计的在网计算方法受交换机等硬件设备限制, 目前最大只支持 256 字节的数据计算, 长报文的通信优化功能需进一步扩展.

本文方法的相关代码已开源于 Github (https://github.com/pplab/openmpi-inc).

致谢

感谢中国科学技术大学超级计算中心与华为技术有限公司南京研究所对本文工作的大力支持.

参考文献

1 Walker DW, Dongarra JJ. MPI: A standard message passing interface. Supercomputer, 1995, 12(1): 56–68.

2 Thakur R, Rabenseifner R, Gropp W. Optimization of collective communication operations in MPICH. The International Journal of High Performance Computing Applications, 2005, 19(1): 49–66. [doi: 10.1177/1094342005051521]

3 Vadhiyar SS, Fagg GE, Dongarra J. Automatically tuned collective communications. Proceedings of the 2000 ACM/IEEE Conference on Supercomputing. Dallas: IEEE, 2000. 3.

4 Parker S, Chunduri S, Harms K, *et al*. Performance evaluation of MPI on Cray XC40 Xeon Phi systems. Cray User Group Proceedings. Stockholm: CUG, 2018. 1–7.

5 Laguna I, Marshall R, Mohror K, *et al*. A large-scale study of MPI usage in open-source HPC applications. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. Denver: ACM, 2019. 31.

6 Chunduri S, Parker S, Balaji P, *et al*. Characterization of MPI usage on a production supercomputer. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. Dallas: IEEE, 2018. 386–400.

7 梁润秋, 沈瑜, Alhusaini N, 等. 超算中心典型科研应用特征统计与分析. 小型微型计算机系统, 2022, 43(6): 1121–1127.

8 Hoefler T, Squyres JM, Rehm W, *et al*. A case for non-blocking collective operations. Proceedings of 2006 International Symposium on Parallel and Distributed Processing and Applications. Sorrento: Springer, 2006. 155–164.

9 NVIDIA. NCCL library. https://github.com/NVIDIA/nccl. (2018-06-18)[2022-02-28].

10 Jia XY, Song ST, He W, *et al*. Highly scalable deep learning training system with mixed-precision: Training ImageNet in four minute. arXiv: 1807.11205, 2018.

11 Mikami H, Suganuma H, U-chupala P, *et al*. ImageNet/ResNet-50 training in 224 seconds. arXiv: 1811.05233, 2018.

12 Ying C, Kumar S, Chen DH, *et al*. Image classification at supercomputer scale. arXiv: 1811.06992. 2018.

13 Barnett M, Littlefield R, Payne DG, *et al*. Global combine on mesh architectures with wormhole routing. Proceedings Seventh International Parallel Processing Symposium. Newport: IEEE, 1993. 156–162.

14 Liu VW, Chen CY, Chen RB. Optimal all-to-all personalized exchange in *d*-nary banyan multistage interconnection networks. Journal of Combinatorial Optimization, 2007, 14(2): 131–142.

15 Petrini F, Coll S, Frachtenberg E, *et al*. Hardware- and software-based collective communication on the Quadrics network. Proceedings of IEEE International Symposium on Network Computing and Applications. Cambridge: IEEE, 2001. 24–35.

16 Gara A, Blumrich MA, Chen D, *et al*. Overview of the Blue Gene/L system architecture. IBM Journal of Research and Development, 2005, 49(2–3): 195–212.

17 Arimilli B, Arimilli R, Chung V, *et al*. The PERCS high-performance interconnect. Proceedings of the 18th IEEE Symposium on High Performance Interconnects. Mountain View: IEEE, 2010. 75–82.

18 Mai L, Rupprecht L, Alim A, *et al*. NetAgg: Using middleboxes for application-specific on-path aggregation in data centres. Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies. Sydney: ACM, 2014. 249–262.

19 Alverson B, Froese E, Kaplan L, *et al*. Cray® XC™ series network. Cray Inc., White Paper WP-Aries01-1112, 2012.

20 Graham RL, Bureddy D, Lui P, *et al*. Scalable hierarchical aggregation protocol (SHArP): A hardware architecture for efficient data reduction. Proceedings of 2016 1st International Workshop on Communication Optimizations in HPC. Salt Lake City: IEEE, 2016. 1–10.

21 Graham RL, Levi L, Burredy D, *et al*. Scalable hierarchical aggregation and reduction protocol (SHARP)™ streaming-aggregation hardware design and evaluation. Proceedings of the 35th International Conference on High Performance Computing. Frankfurt: Springer, 2020. 41–59.

22 Ramesh B, Suresh KK, Sarkauskas N, *et al*. Scalable MPI collectives using SHARP: Large scale performance evaluation on the TACC frontera system. Proceedings of 2020 Workshop on Exascale MPI (ExaMPI). Atlanta: IEEE, 2020. 11–20.

23 InfiniBand Trade Association. http://www.infinibandta.org. (2017-05-24)[2022-02-28].

(校对责编: 孙君艳)