计算机研究与发展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202220684
61(2): 503−517, 2024

硬件集合通信中聚合树构建方法

陈淑平　尉红梅　王　飞　李　祎　何王全　漆锋滨
（国家并行计算机工程技术研究中心　北京　100190）

Method to Create Aggregate Tree for Hardware Supported Collectives

Chen Shuping, Wei Hongmei, Wang Fei, Li Yi, He Wangquan, and Qi Fengbin
（*National Research Center of Parallel Computer Engineering & Technology*, *Beijing* 100190）

Abstract　Traditional MPI (message passing interface) collectives are implemented by point-to-point messages, and have poor performance. Hardware supported collectives have attracted more and more attention due to their high performance and low CPU utilization. Aggregate tree has crucial impact on the performance of hardware supported collectives. We study the factors that affect the performance of hardware supported collectives, and propose a cost model for hardware supported collectives and an efficient method to create aggregate trees. The method includes three parts. Firstly, we choose appropriate aggregate tree type and breadth according to the operation type and the size of aggregate messages to do tradeoff between network transmission time and data processing time. Secondly, we propose a method to create hierarchical minimum height aggregate tree of type Ⅰ, which reduces the number of inter-group communication. Thirdly, we put forward a method to create the minimum cost aggregate tree of type Ⅱ, which minimizes the number of used switches. In the Sunway interconnection network, we test the proposals. In the presence of network noise, the message latency of the hierarchical minimum height aggregate tree of type Ⅰ is reduced by 24%−89% compared with that of the traditional method. The aggregate entries used by the minimum cost aggregate tree of type Ⅱ for typical communication patterns are reduced by 90% compared with that of the traditional method.

Key words　MPI collectives; aggregate tree; cost model; high-speed interconnect; high-performance computing

摘　要　传统的 MPI (message passing interface) 集合通信是基于点到点消息实现的，性能较低；而硬件集合通信具有性能高、CPU 占用率低等优点，正受到越来越多的关注. 硬件集合通信中，聚合树对集合通信性能具有至关重要的影响. 研究了影响硬件集合通信性能的因素，提出了硬件集合通信开销模型，并以此为基础提出了构建硬件集合通信聚合树的方法. 该方法主要包括 3 个部分：1）根据操作类型、聚合数据包大小等确定聚合树类型及聚合树宽度，从而在网络传输开销与数据计算开销之间取得平衡；2）提出了最小高度分层 k 项Ⅰ型聚合树构建方法，降低了跨组聚合包的个数；3）提出了构建最小代价Ⅱ型聚合树的方法，减少所使用的交换机数量. 在神威互连网络中对聚合树构建方法进行了全面测试，当存在网络噪声的情况及分层 k 项Ⅰ型聚合树构建方法下的消息延迟相比传统构建方法下降了 24%~89%；典型通信模式时，最小代价Ⅱ型聚合树使用的交换机聚合条目数相比优化前下降了约 90%.

关键词　MPI 集合通信；聚合树；开销模型；高速互连；高性能计算

中图法分类号　TP393

收稿日期：2022-08-08；**修回日期**：2023-04-06

基金项目：国家重点研发计划项目 (2020YFB0204602)
This work was supported by the National Key Research and Development Program of China (2020YFB0204602).

通信作者：漆锋滨（qifb116@sina.com）

目前超级计算机已经迈入E级计算时代，但随着摩尔定律接近失效、颠覆性使能技术迟迟不能实现，超级计算机在访存、通信、能耗等方面仍面临严重的发展瓶颈[1]. 而MPI(message passing interface)集合通信[2]是高性能计算中最重要的通信类型，对整机系统的性能具有重要的影响，提升MPI集合通信性能是突破通信墙的重要手段. 传统的MPI集合通信是基于点到点消息实现的[3-4]，随着系统规模的不断扩大，这种实现方式面临越来越严重的性能及可扩展性问题，而硬件集合通信具有性能高、可扩展性好等优点，正受到越来越多的关注[5].

硬件集合通信中，数据沿聚合树进行规约或广播操作，聚合树的高度、负载均衡性等对集合操作的性能具有至关重要的影响. 本文研究了影响硬件集合通信性能的因素，提出了硬件集合通信开销模型，并以此为基础提出了构建硬件集合通信聚合树的方法. 本文主要贡献包括4个方面：

1) 提出了硬件集合通信开销模型，能够较精确地计算硬件集合通信的时间开销；

2) 研究了如何根据消息大小确定聚合树的宽度，从而在网络传输开销与数据计算开销之间取得平衡，以降低集合操作的延迟；

3) 针对Ⅰ型聚合树，提出了最小高度分层k项聚合树构建方法，相比传统的Ⅰ型聚合树，降低了跨组聚合包的个数，减少了网络拥塞；

4) 针对Ⅱ型聚合树，提出了最小代价Ⅱ型聚合树构建方法，减少了聚合树使用的交换机数量.

1 研究背景

1.1 硬件集合通信简介

近年来，网络计算技术发展迅速[6-7]. 网络计算可将数据规约计算、协议处理、数据加解密等原本由处理器承担的计算任务卸载到网络设备中进行，解放了CPU，使CPU可以专注于其他任务的处理. 在高性能计算领域，可利用网络计算提升MPI集合通信性能. 本文将基于网络计算实现的MPI集合通信称为硬件集合通信. MPI集合通信中，Barrier, Bcast, Reduce, Allreduce是应用最广泛的操作，它们均是1对多或多对1的通信模式，适于以硬件集合通信方式实现. 图1显示了Allreduce的实现过程，它利用聚合树完成数据聚合及结果分发操作，该树中叶节点对应通信进程，根节点及中间节点对应交换机. 进行集合通信时，数据先从叶节点向根节点聚合，并在聚合的过程中计算规约结果；到达根节点后再沿反方向将规约结果广播给所有叶节点. 其他集合操作的实现方式与此类似.

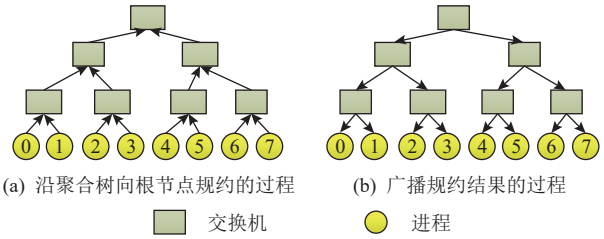


Fig. 1 Implementation phases of Allreduce operation based on in-network-computing

图1 基于网络计算的Allreduce操作实现过程

目前，很多公司推出了面向高性能计算的硬件集合通信技术，如IBM BlueG/Q中的硬件集合通信原语[8-11]、Mellanox的SHARP技术[5, 12-14]、神威互连网络中的硬件集合通信[15]等. 相较于点到点消息实现的集合通信，硬件集合通信的性能有了显著提升. Zimmer等人[16]在Summit系统中利用基准程序对SHARP性能进行了测试，4 096个节点规模下集合消息性能至少提升了1倍.

1.2 硬件集合通信使用中存在的问题

硬件集合通信除了需要特殊的硬件支持外，还需要网络管理软件的密切配合，以完成聚合树分配、多播路由配置等操作. 其中，聚合树分配是其中最为关键的操作. 现有的硬件集合通信技术在分配聚合树时还存在3个问题：

1) 缺乏精确的集合通信开销模型，无法更好地指导聚合树构建. 基于α-β模型的软件集合通信开销模型[3]过于简化，而针对特定网络的专用模型又不具备通用性[17]，它们均不适用于硬件集合通信.

2) 构造聚合树时未考虑相互干扰问题. 当系统中存在大量通信域时，需要为每个通信域分配1棵独立的聚合树，以降低各通信域间的相互干扰，而现有的聚合树分配方法未充分考虑到该问题. 例如，SHARP技术中每个作业的所有通信域共享同一棵聚合树[18]，这会产生严重的相互干扰. 现有的聚合树构建方法也没有考虑到同一聚合树内不同消息间的相互干扰问题. 例如，传统方法构建出的聚合树会产生较多的跨组通信，使得同一集合操作的不同消息因竞争通信链路而相互干扰.

3) 未考虑硬件通信资源不够用问题. 网络中用于支持硬件集合通信的资源是有限的，而现有的聚合树构建方法未考虑该问题，导致出现通信资源不够用的情况. 例如，BlueG/Q中每个网络节点最多支

持 12 棵聚合树，在很多情况下，由于聚合树数量不足，无法使用硬件集合通信[11].

针对上述 3 个问题，本文研究如何建立硬件集合通信开销模型，以及如何基于该模型高效地构建聚合树.

2 硬件集合通信开销模型

2.1 定义及假设

定义 1. 互连网络. 互连网络定义为有向图 $I=(N, L)$，其中 N 为网络节点集合，L 为通信链路集合. 网络节点包括网卡及交换机. 本文假设网卡在 I 中均是叶节点.

定义 2. 聚合树. 在互连网络 $I=(N, L)$ 上目标集合 $M(M\subset N)$ 对应的聚合树是一棵树 $A=(N_A, L_A)$，$M\subseteq N_A\subseteq N$，其中 M 是通信域中各进程所在的网卡集合. A 中边 (μ,υ) 的长度定义为 I 中 μ 到 υ 的距离，也即 I 中从 μ 到 υ 经过的网络链路数. A 不必是 I 的子树，也即不要求 $L_A\subseteq L$. 另外，M 中允许有重复元素，从而使得每个节点上可同时运行多个通信进程. 聚合树还需满足度约束条件，也即每个节点的最大子节点数不能超过硬件阈值.

定义 3. 聚合树高度. 设 $A=(N_A, L_A)$ 是互连网络 $I=(N, L)$ 上目标集合 M 对应的一棵聚合树，A 中从叶节点出发到达根节点所经过的边的最大条数，称为聚合树的高度. 只有 1 个节点的聚合树高度为 0.

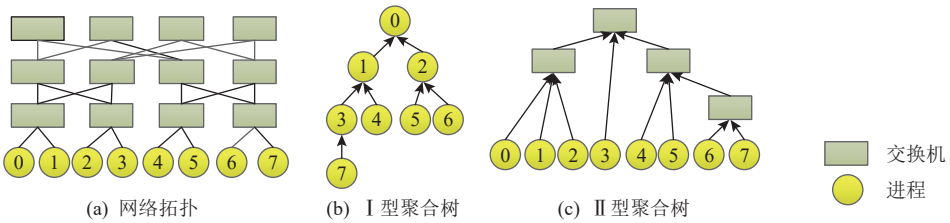
定义 4. 聚合树半径. 设 $A=(N_A, L_A)$ 是互连网络 $I=(N, L)$ 上目标集合 M 对应的一棵聚合树，A 中从叶节点出发到根节点所经过边的长度之和的最大值称为聚合树半径. 聚合树半径实际上是从各叶节点出发沿该聚合树依次经过各祖先最终到达根节点时所经过的最大链路条数.

现有的硬件集合通信技术中，有的仅支持将集合操作卸载到网卡中进行，有的则还可将集合操作卸载到交换机中进行. 根据不同的卸载位置，本文将聚合树分为 Ⅰ 型聚合树及 Ⅱ 型聚合树 2 种类型.

定义 5. Ⅰ 型聚合树. 设 $A=(N_A, L_A)$ 是互连网络 $I=(N, L)$ 上目标集合 M 对应的一棵聚合树，若 $M=N_A$，则称 A 为 Ⅰ 型聚合树. 该类聚合树中，集合操作均被卸载到网卡中执行，且仅使用进程所在的那些网卡.

定义 6. Ⅱ 型聚合树. 设 $A=(N_A, L_A)$ 是互连网络 $I=(N, L)$ 上目标集合 M 对应的一棵聚合树，$T=(N_T, L_T)$ 是 I 的一棵子树，$M\subseteq N_A\subseteq N_T\subseteq N$，且 T 中叶节点均在 M 中. 若 $\forall u\in N_A$，u 在 A 中的父亲 p 也是 u 在 T 中的某个祖先，则称 A 为 Ⅱ 型聚合树. Ⅱ 型聚合树中的集合操作均被卸载到交换机中进行.

图 2 显示了 Ⅰ 型聚合树和 Ⅱ 型聚合树. 假设有 8 个进程参与集合通信，每个网卡上 1 个进程，互连网络的拓扑结构如图 2(a)所示；图 2(b)是一棵 Ⅰ 型聚合树，高度为 3，半径为 12；图 2(c)是一棵 Ⅱ 型聚合树.



(a) 网络拓扑　(b) Ⅰ 型聚合树　(c) Ⅱ 型聚合树

Fig. 2 Aggregate trees of type Ⅰ and type Ⅱ

图 2 Ⅰ 型聚合树和 Ⅱ 型聚合树

2.2 开销模型

本文采用式(1)~(4)所示的开销模型，模型中各变量的含义如表 1 所示. 本文关注于如何降低聚合过程的开销，下面着重对聚合过程开销进行说明.

$T_{\text{aggregate}}$ 是聚合过程的时间，其包括聚合包在网络链路上的传输时间，以及在网络节点内的处理时间. 假设聚合树是 k 叉完全树，聚合树的半径为 r_{atree}，则聚合包在网络链路中的传输总时间为 $r_{\text{atree}}\times l$. 设每个聚合包的匹配处理时间为 θ、存储转发时间为 $\frac{S'}{\beta}$、规约计算时间为 $S^*\times\gamma$，则聚合包在网络节点内的处理总时间为 $h_{\text{atree}}\times k\times\left(\theta+\frac{S'}{\beta}+S^*\times\gamma\right)$. 最终得到式(2)所示的聚合时间开销.

为进一步简化模型，除了假设聚合树是 k 叉完全树之外，还假设任意 2 个进程之间单播路由的路径长度相等，均需经过 δ 条链路，则对 Ⅰ 型聚合树，$h_{\text{atree}}\approx\log_k P$，$r_{\text{atree}}\approx\delta\times\log_k P$，从而得到式(3)所示的聚合过程开销. 而对 Ⅱ 型聚合树，$h_{\text{atree}}\approx\lceil\log_k P\rceil$，$r_{\text{atree}}\approx 0.5\delta$，从而得到式(4)所示的聚合过程开销.

$$T_{\text{coll}}=T_{\text{post_wr}}+T_{\text{fetch_wr}}+T_{\text{fetch_data}}+T_{\text{aggregate}}+T_{\text{bcast}}+T_{\text{write_wc}}+T_{\text{noise}},\quad(1)$$

$$T_{\text{aggregate}}=r_{\text{atree}}\times l+h_{\text{atree}}\times k\times\left(\theta+\frac{S'}{\beta}+S^{*}\times\gamma\right),\tag{2}$$

$$T_{\text{aggregate_I}}\approx\log_k P\times\left[\delta l+k\left(\theta+\frac{S'}{\beta}+S^{*}\times\gamma\right)\right],\tag{3}$$

$$T_{\text{aggregate_II}}\approx\frac{1}{2}\delta l+\lceil\log_k P\rceil\times k\left(\theta+\frac{S'}{\beta}+S^{*}\times\gamma\right).\tag{4}$$

Table 1 Symbols Used in the Cost Model of Hardware Supported Collectives

表 1 硬件集合通信开销模型中使用的符号

符号	含义
T_{coll}	集合操作的总时间开销
$T_{\text{post_wr}}$	应用投递 1 个集合操作描述符的时间
$T_{\text{fetch_wr}}$	网卡读取 1 个集合操作描述符的时间
$T_{\text{fetch_data}}$	网卡从内存读取操作数的时间
$T_{\text{aggregate}}$	聚合过程的时间开销
T_{bcast}	广播过程的时间开销
$T_{\text{write_wc}}$	网卡写完成条目及结果到内存的时间
T_{noise}	噪声导致的开销
S	用户消息大小（不超过 2 KB）
S'	聚合包大小
S^{*}	规约数据长度
β	PCIE 链路/网络链路带宽
l	消息经过 1 条网络链路的时间
h_{atree}	聚合树高度
r_{atree}	聚合树半径
P	进程总数
k	k 叉树的宽度或 k 项树的基
θ	每个聚合包的匹配处理时间
γ	单位长度数据的规约计算时间
δ	任 2 个进程间单播路由的最大跳步数

2.3 模型验证

本文在新一代神威超级计算机上测试了各类集合消息的延迟，并与开销模型计算出的结果进行对比. 实验环境如图 3 所示.

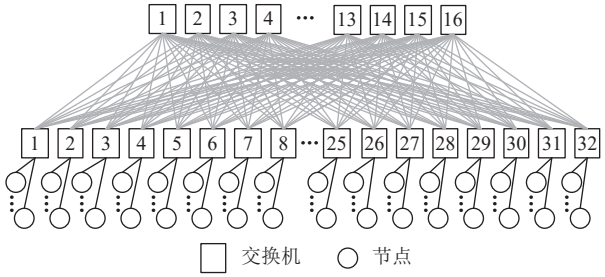


Fig. 3 2-level fat tree used in the test environment

图 3 测试环境使用的 2 层胖树

使用 1 个超节点进行测试，超节点内共有 256 个节点，每个节点安装 1 块消息处理芯片，每块消息处理芯片有 2 个网络端口，故整个超节点内共有 512 个网络端口. 这 512 个网络端口通过一棵 2 层胖树进行互连. 该胖树共有 48 台交换机和其中 32 台叶交换机和 16 台骨干交换机. 每台交换机的端口数均为 40，其中有些端口未被使用. 每台叶交换机有 16 个端口用于连接消息处理芯片的网络接口，16 个端口用于连接骨干交换机.

分别在不同进程数下测试了各种类型、各种长度集合消息的延迟. 测试方法为：每种类型及长度均测试 5 000 次，每次测试进程均记录集合操作的时间开销，取耗时最长的那个进程的时间开销作为该次集合通信的时间开销，再取这 5 000 个时间开销的中位值作为当前类型、当前长度下集合消息的延迟.

限于篇幅，本文仅显示了 I 型聚合树的部分测试结果. 不同进程数下，长度为 8 B 的广播操作的理论计算延迟与实际测试延迟如图 4 所示；进程数为 512 时不同长度 Bcast 消息的理论计算延迟与实际测试延迟如图 5 所示. 可以看出，I 型聚合树的理论计算延迟与实际测试延迟最多相差 0.5 μs. 这表明本文提出的硬件集合通信开销模型能够较准确地拟合实测延迟.

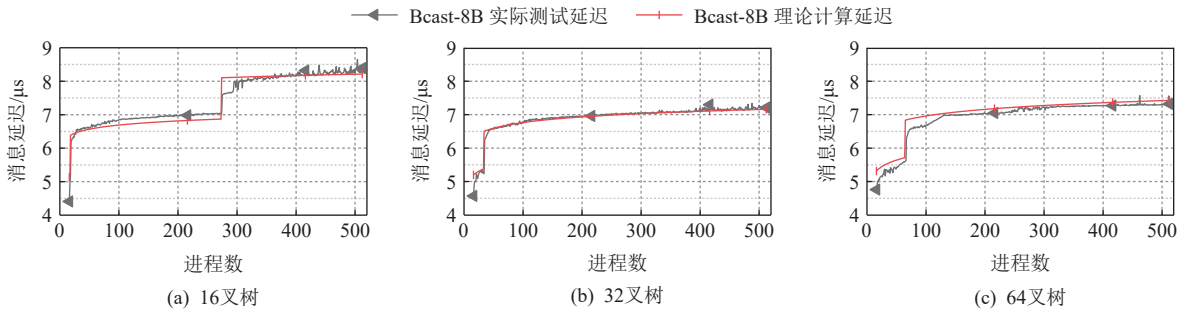
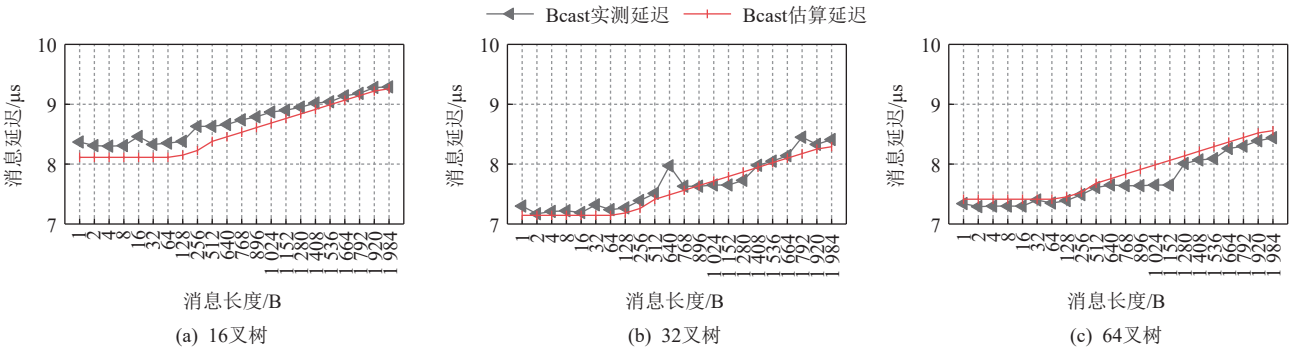


Fig. 4 Measured latency and estimated latency of collectives for different count of processes

图 4 不同进程数下集合操作的实测延迟及理论计算延迟



(a) 16叉树 (b) 32叉树 (c) 64叉树

Fig. 5 Measured latency and estimated latency of collectives for different message size when process count is 512

图 5 进程数为 512 时不同消息大小下集合消息的实测延迟及理论计算延迟

3 Ⅰ型聚合树的构建

给定任意 P 个进程，为它们构造一棵开销最小的Ⅰ型聚合树是非常困难的，因为不但要考虑聚合树宽度、聚合树半径等因素，还要考虑通信与通信重叠、通信与计算重叠、消息间相互干扰等因素的影响，故目前Ⅰ型聚合树均是通过启发式算法进行构造. 本节首先研究如何根据消息类型、消息长度等确定Ⅰ型聚合树的宽度 k，然后研究如何快速构造一棵聚合树，使得跨组聚合包的个数尽可能地少，尽量降低消息间的相互干扰.

3.1 根据聚合包大小确定聚合树的宽度

聚合过程的时间开销包括聚合包的传输时间以及处理时间. 聚合树的宽度 k 对聚合过程的时间开销具有重要影响. 直观上看，如果 k 很小，则聚合树很高，相应地聚合包的传输开销就很大；如果 k 很大，则每个网卡需要处理的聚合包很多，相应地聚合包的处理开销就很大. 需要研究 k 对聚合时间开销的影响.

先看如何选择 k 才能使式(3)计算出的聚合开销 $F(k)$ 最小. 记 $\delta\times l=\alpha$，$\theta+\frac{S^{'}}{\beta}+S^{*}\times\gamma=b$，定义如式(5)所示的 $F(k)$，其导数函数如式(6)所示.

$$F(k)=\log_k P\times\left[\delta\times l+k\times\left(\theta+\frac{S'}{\beta}+S^{*}\times\gamma\right)\right]=\log_k P\times[a+kb], \tag{5}$$

$$F'(k)=\frac{(\ln P)[kb(\ln k-1)-a]}{k\ln^2 k}. \tag{6}$$

使 $F(k)$ 取最小值的 k 记为 k^{*}，此时的 $F(k)$ 记为 F^{*}. 根据式(5)(6)，要使 $F(k)$ 取最小值，需 $F'(k)=0$，也即 $k(\ln k-1)=\frac{a}{b}$，而函数 $f(k)=k(\ln k-1)$ 是单调递增的. 对不同的集合操作类型及消息长度，$\frac{a}{b}$ 取值不同，使 $F(k)$ 取最小值的 k 也不同. 在本文的实验环境中，对同步、广播操作来说，$k=41$ 时 $F(k)$ 取最小值.

很多情况下，选择其他 k 值时，$F(k)$ 与 F^{*} 差别不大. 以同步、广播操作为例，在本文的实验环境中，设 $P=512$，$a=1.12$ μs，$b=0.01$ μs，$F(k)$ 的曲线如图 6 所示，可以看出，当 $24\leqslant k\leqslant 64$ 时，$F(k)$ 相差不大，不超过 0.1 μs. 对每种类型、每种长度的集合消息，都可以确定区间 $[k_{\min},k_{\max}]$，使得 $\forall k\in[k_{\min},k_{\max}]$ 时，$|F(k)-F^{*}|\leqslant\varepsilon$ 成立(ε 是常数). 本文测试环境中使用的 k 如表 2 所示(仅显示了部分结果). 可以发现，随着聚合包长度增大，$k_{\min}$ 及 $k_{\max}$ 均逐渐下降. 对该现象的直观解释是：当聚合包长度较小时，聚合包在网络

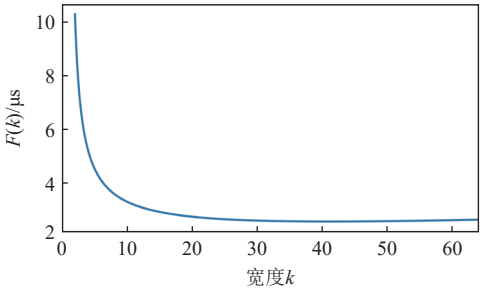


Fig. 6 Curve of $F(k)$ for Barrier and Bcast operation

图 6 同步、广播操作的 $F(k)$ 曲线

Table 2 Type Ⅰ Aggregate Tree Breadth Used in the Test Environment

表 2 测试环境中使用的Ⅰ型聚合树宽度

消息类型	聚合包长度/B	$k_{\min}$	$k_{\max}$	推荐k
Barrier/Bcast	0	32	64	32
Reduce/AllReduce	32	16	63	32
Reduce/AllReduce	64	15	53	32
Reduce/AllReduce	128	13	40	16
Reduce/AllReduce	256	10	28	16
Reduce/AllReduce	512	8	19	16
Reduce/AllReduce	1 024	6	12	8
Reduce/AllReduce	2048	5	8	8

链路上的传输时间在总聚合时间中占比最大,此时应该采用较大的 k,从而使得聚合包在网络链路上的传输尽量并行;而当聚合包长度较大时,聚合包在网卡内的处理时间在总聚合时间中占比最大,此时应采用较小的 k,以使更多的网卡参与聚合包的处理.

3.2 传统的聚合树构建方法

确定 k 值后就可以构建聚合树了.式(3)假设任意 2 个进程间的单播路由经过的链路数相等,但实际情况要复杂得多.使用相同的 k 值构造的 2 棵聚合树值半径 r_{atree} 可能不同,导致聚合包的传输开销不同.带度约束的最小半径聚合树问题是 NP 难的[19-21],尚未有高效的求解办法.本文目标不是构造最优聚合树,而是快速构造出一棵满足度约束条件、半径尽可能小、冲突尽可能少的聚合树.本节首先介绍 2 种传统的聚合树构建方法.

3.2.1 k 叉树

定义 7. k 叉树.k 叉树是一棵完全树,除最后一个中间节点外,根节点及其他中间节点都有 k 个子节点,故 k 叉树是平衡树,每个中间节点处理的聚合包个数相等.

在 k 叉树中,按广度优先遍历方式为进程编号,根进程编号为 0.若进程在通信域内的编号为 my_rank,则其父节点编号为 $(my_rank-1)/k$,第 i 个子节点编号为 $my_rank\times k+i(i\geqslant 1)$.

由于未考虑网络拓扑结构,在胖树网络[22-23]、蜻蜓网络[24]等具有分组结构的网络中,k 叉树会产生大量的组间通信,导致网络拥塞.16 个进程构建聚合树,其网络拓扑如图 7(a)所示,4 叉树如图 7(b)所示.粗线表示该聚合包需要经过最顶层的交换机,图 7(b)中共有 8 个聚合包经过最顶层的交换机.如果最顶层的交换机传输能力不足,则很容易在顶层交换机处形成拥塞.而图 7(c)是另一棵聚合树,其高度、半径均与图 7(b)所示聚合树一样,但仅有 2 个经过顶层交换机的聚合包,因而具有更优的性能.

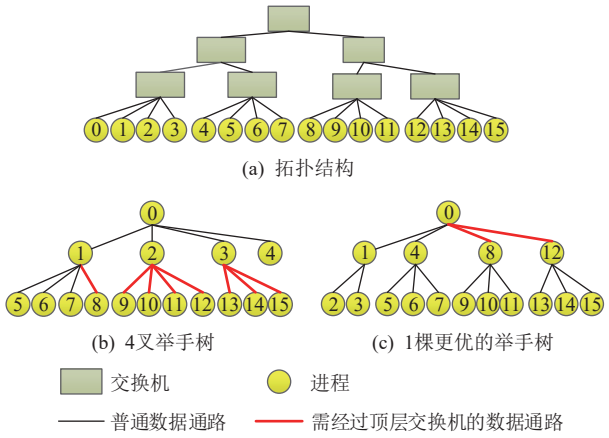


Fig. 7 k-ary aggregate tree is prone to generate lots of inter-group communication

图 7 k 叉树易产生大量的组间通信

3.2.2 k 项树

定义 8. k 项树.k 项树[25]是一组递归定义的树 T_n,其中 T_0 是 1 个单节点的树;而 T_n 是由 k 棵 T_{n-1} 连接起来构成的树,其中 $k-1$ 棵 T_{n-1} 的根节点直接连接到另外那棵 T_{n-1} 的根节点上.k 项树第 i 层的节点数是 $(k-1)^i\times C_n^i$,节点总数是 k^n,高度为 n.

图 8 分别显示了一棵 2 项树及一棵 4 项树.2 项树是特殊的 k 项树,在软件集合通信中已得到了广泛使用,例如在 MPICH 中短的 Bcast、Reduce 均是通过 2 项树实现的[3];而大于 2 的 k 项树目前使用得较少.

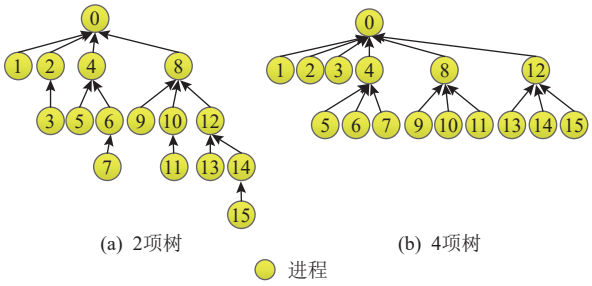


Fig. 8 k-nomial tree (k=2, 4)

图 8 k 项树(k=2, 4)

与 k 叉树不同,k 项树中不同深度的节点拥有不同数量的子节点,其中根节点的子节点数最多,为 $(k-1)\times\lceil\log_k P\rceil$;而随着深度的增加,子节点数越来越少,故 k 项树是非平衡树.

由于 k 叉树是平衡树,k 叉树中上层节点要等待下层节点处理完毕之后才开始处理聚合包,故上层节点会存在空等待现象.而 k 项树是非平衡树,位于不同层的节点可以并发处理聚合包,不存在空等待现象.

在 k 项树中,按深度优先遍历方式为进程编号.进程编号用 k 进制表示为 $k_{n-1}\cdots k_2,k_1,k_0$,其中最低位共有 i 个 $0(0\leqslant i\leqslant n)$,则第 i 个位置 0 即可得到其父节点编号,从最低的 i 位中任选一位置为 $1,2,\cdots,k-1$,则得到其子节点的编号最多有 $i\times(k-1)$ 个子节点.

每个进程调用算法 1 所示的 $build_k_nomial_tree$ 函数计算 $parent_rank$ 及 $children_cnt$.与 k 叉树一样,k 项树也会产生很多跨组通信.

算法 1. $build_k_nomial_tree$.

输入:本进程编号 my_rank,基数 k,进程总数 P;

输出:父节点 $parent_rank$,子节点数 $children_cnt$,

子节点列表 *children*[].

① *mask*=1, *children_cnt*=0;
② while(*mask*<*P*)
③ if(*my_rank*%(*k*×*mask*)) then
④ *parent_rank* = *my_rank*/(*k*×*mask*)×(*k*×*mask*);
⑤ break;
⑥ end if
⑦ *mask*=*mask*×*k*;
⑧ end while
⑨ *mask*=*mask*/*k*;
⑩ while(*mask*>0) do
⑪ for (*r*=1; *r*<*k*; *r*++)
⑫ *child*=*my_rank*+*mask*×*r*;
⑬ if (*child* < *P*) then
⑭ *children*[*children_cnt*]=*child*;
⑮ *children_cnt*+ =1;
⑯ end if
⑰ end for
⑱ *mask*=*mask*/*k*;
⑲ end while
⑳ 返回 *parent_rank*, *children_cnt*, *children*.

3.3 分层 *k* 项树构建方法

传统的聚合树构建方法未考虑网络拓扑结构，容易产生较多的组间通信，从而造成网络拥塞. 为解决此问题，本文提出了分层 *k* 项树构建方法.

分层思想是优化集合通信的一种重要策略[26-29]，该策略利用了不同的层具有不同数据传输性能的特点，将同一层内的进程组织成组，使尽可能多的通信在组内完成，以降低层间通信、提升集合消息性能.

在构建 I 型聚合树时，利用分层思想也可以降低聚合过程的时间开销. Zhao 等人[25] 提出了一种简单方法：先将进程分为不同的组，并为每个组单独构建一棵 *k* 项树；然后将各组的首进程组成一个新的组，并创建一棵 *k* 项树，从而将所有组连接起来形成一棵更大的 *k* 项树. 进行聚合操作时，同一个组内的进程会先聚合到本组的首进程，首进程再聚合到更高层的首进程，从而大大减少组间通信. 但该方法产生的 *k* 项树可能具有较大的高度. 图 9 举了一个例子：通信域内共有 14 个进程，依图案分属 4 个不同的组. 当 *k* = 2 时，构建的聚合树如图 9(a)所示，其高度为 4. 实际上，可以使某个组的首进程连接到另一个组的非首进程下，从而使得聚合树更加平衡，以降低聚合树的高度. 图 9(b)即是一棵更优的聚合树，其高度为

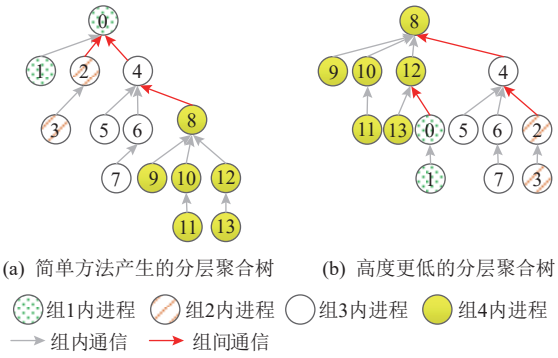


Fig. 9 Aggregate tree with hierarchical structure

图 9 具有分层结构的聚合树

3. 本文的目标是在最小化组间通信的同时使聚合树高度最低.

3.3.1 问题定义

本文将具有相同通信特性的一组进程称为一个进程组. 例如，位于同一节点内的多个进程，以及位于同一交换机内的多个进程等，均可构成一个进程组. 进程组可以嵌套，从而使得一个进程组可包含多个子进程组，最终构成 1 个层次结构.

进程组由如图 10 所示的数据结构定义. 每个进程组都对应一棵 *k* 项树，按深度优先方式为 *k* 项树内的节点编号. 进程被映射到 *k* 项树的某个节点上，树中有些节点可能无映射进程. 该数据结构中，*cnt* 是 *k* 项树的节点数，它必须是 *k* 的幂；*ranks_id* 是 *k* 项树内各节点对应的进程号，*nodes_id* 是各进程对应的 *k* 项树节点号；*sub_groups* 定义了属于该进程组的子进程组，共有 *sub_grp_cnt* 个子进程组.

```
struct process_group{
    int cnt;
    int ranks[];
    int nodes[];
    int sub_grp_cnt;
    struct process_group  sub_groups[];
}
```

Fig. 10 The data structure of the process group

图 10 进程组的数据结构

下面探讨最小高度分层 *k* 项 I 型聚合树问题. 给定 *m* 个互不相交的进程组 $g_0, g_1, \cdots, g_{m-1}$，*g* 是满足 2 个条件的进程组：1) *g* 包含 $g_0, g_1, \cdots, g_{m-1}$ 的所有进程；2) *g* 对应的 *k* 项树中，除首进程外其余进程的父节点均位于 g_i 内；3) *g* 对应的 *k* 项树中，每个节点的儿子数不超过硬件限制. 该问题为从中找出具有最小高度的进程组 *g*，从而为进程组 $g_0, g_1, \cdots, g_{m-1}$ 构建一棵高度最小的分层 *k* 项 I 型聚合树. 进程组 $g_i (0 \leqslant i < m)$

内嵌套的子进程组也需满足条件2.

设g_i^0是进程组g_i首进程的进程号,根据k项树的性质,条件2等价于$g.nodes\left[g_i^0\right]\%(g_i.cnt)=0$且对$g_i$内每个进程$g_i^j$有$g.nodes\left[g_i^j\right]/(g_i.cnt)=g.nodes\left[g_i^0\right]/(g_i.cnt)$,也即进程$g_i^0$被映射到某个编号为$g_i.cnt$倍数的聚合树节点内,其余进程依次映射到后续节点内.

3.3.2 构建方法

给定m个进程组$g_0,g_1,\cdots,g_{m-1}$,且每个g_i均已构建好各自的k项树之后,即完成了*ranks*的设置,采用算法2为这些进程组构建一棵更大的k项聚合树.算法2采用贪心策略,每次找出2个最大的进程组g_0和g_1,然后在g_0的空闲子树内寻找能够容纳g_1的位置.如果找不到这样的位置,则将g_0的聚合树高度加1,即*cnt*变为原来的k倍,从而使g_0内有足够的空余位置容纳g_1.该过程持续下去,直到最终只剩1个进程组为止.算法2行⑤代码用于为$g[i]$寻找可用的位置,仅需检查分块的第1个位置是否可用即可判断整个分块是否可用.行⑫~⑭代码用于将$g[i]$的*ranks*映射表拷贝到新的聚合树中.

算法2. *merge_groups*.

输入:子进程组列表g[]及列表长度m,基数k,父进程组*parent*;

输出:无.

① 将$g[0],g[1],\cdots,g[m-1]$按照*cnt*字段由大到小的顺序排列;
② $parent.cnt=g[0].cnt$;
③ for(i=0; i<m; ++i)
④ for(j=0; j<$parent.cnt$; j+=$g[i].cnt$)
⑤ if($parent.ranks\left[j\right]==-1$)then
⑥ break;
⑦ end if
⑧ end for
⑨ if($j\geqslant parent.cnt$) then
⑩ $parent.cnt=pareng\times k$;
⑪ end if
⑫ for(x=0; x<$g[i].cnt$; ++x)
⑬ $parent.ranks\left[j+x\right]\leftarrow g[i].ranks[x]$;
⑭ end for
⑮ end for

采用数学归纳法可证明算法2所产生的k项树是高度最小的分层k项Ⅰ型聚合树,限于篇幅,不再赘述证明过程.

算法3以递归方式为每个层次建立进程到聚合树节点的映射关系.算法4根据拓扑信息建立分层的k项树.*build_hierarchical_tree*建立的聚合树中,每个组的进程均位于同一子树内,从而保证除该组的首进程外其余进程均向本组内的进程发送聚合包,故跨组聚合包的个数相比传统聚合树构建方法减少了.在使用算法4时,需要选择合适的k,以保证聚合树内每个节点的儿子数不超过硬件限制.

算法3. *build_rank_mapping*.

输入:进程组g;

输出:无.

① if($g.sub_grp_cnt==0$)
② return;
③ end if
④ for(i=0; i<$g.sub_grp_cnt$; ++i)
⑤ *build_rank_mappin*($g.sub_groups[i]$);
⑥ end for
⑦ *merge_groups*($g.sub_groups, g.sub_grp_cnt, k, g$).

算法4. *build_hierarchical_tree*.

输入:本进程编号*my_rank*,基数k,进程总数P;

输出:父节点*parent_rank*,子节点数*children_cnt*.

① 根据网络拓扑信息构建进程组信息,其中底层的组其*sub_grp_cnt*=0,*ranks*填充为该组内各进程的*rank*号,其他层的组需填充*sub_groups*,最高层的组记为g;
② *build_rank_mapping*(g);
③ 遍历$g.ranks$找到*my_rank*在该数组内的下标*node_id*;
④调用*build_k_nomial_tree*($node_id, k, g.cnt$)获得*parent_node_id*、子节点列表*children*及列表长度*cnt*;
⑤ *parent_rank*=$g.ranks$[*parent_node_id*];
⑥ *children_cnt*=0;
⑦ for (i=0; i<*cnt*; i++)
⑧ if ($g.ranks[children[i]]\neq-1$) then
⑨ *children_cnt*+=1;
⑩ end if
⑪ end for
⑫ 返回*parent_rank*及*children_cnt*.

4 Ⅱ型聚合树的构建

本节研究构造Ⅱ型聚合树的方法.构造Ⅱ型聚合树时需要综合考虑网络拓扑、进程位置等因素.另外,在每个交换机内,每棵Ⅱ型聚合树都占用1个聚

合树条目，而硬件支持的聚合树条目是有限的，故创建Ⅱ型聚合树时，还需要考虑是否有可用的聚合树条目.

构造Ⅱ型聚合树时要考虑3个目标：1)最小化聚合树的半径，以降低集合操作延迟；2)尽量让不同的聚合树分布到不同链路上，以降低聚合树间的相互干扰；3)使用尽可能少的聚合树条目，以支持创建更多的Ⅱ型聚合树.

本文分2个步骤构造Ⅱ型聚合树. 1)选择1个交换机作为树根，并构造1棵生成树(称为物理树)；2)对物理树进行裁剪，去掉不需要的交换机.

定义 9. 物理树. 设 M 是某通信域内进程所在的网卡集合，$T=(N_T, L_T)$ 是互连网络 $I=(N, L)$ 的一棵子树，若 T 满足2个条件，则称 T 是关于 M 的物理树：

1) $M\subseteq N_T$;

2) T 的叶节点除 M 中网卡之外，不包含任何其他网卡及交换机.

定义 10. 由物理树推导出的Ⅱ型聚合树. 设 K 是交换机节点的度约束函数，给定互连网络 $I=(N, L)$ 上一棵关于 M 的物理树 $T=(N_T, L_T)$，由物理树 T 导出的Ⅱ型聚合树是满足2个条件的树 $A=(N_A, L_A)$：

1) $M\subseteq N_A\subseteq N_T$;

2)若 $\forall u\in N_A$，u 在 A 中的父亲 p 也是 u 在 T 中的某个祖先；

3) A 中每个交换机 sw 的子节点数不大于度约束 $K(sw)$，其中 $K(sw)$ 大于 sw 在 T 中的子节点数.

实际上，由物理树 T 导出的Ⅱ型聚合树 A 可看成是将 T 中某些节点提升位置而得到的(仅能提升为其祖先的儿子).

定义 11. Ⅱ型聚合树的代价. 设 $A=(N_A, L_A)$ 是互连网络 $I=(N, L)$ 上目标集合 M 对应的一棵Ⅱ型聚合树，A 中交换机的个数称为该Ⅱ型聚合树的代价，记为 $cost(A)$.

4.1 选择树根并构建物理树

树根的位置决定了聚合树的半径，应该选择到通信域内各进程最大跳步数最小的交换机作为树根.

本文采用集中式方法构建物理树. 在该方法中，由集中式的网络管理软件选择树根并构建物理树. 网络管理软件维护了网络的状态信息，包括交换机内空闲的聚合树条目数、网络链路的负载情况等. 网络管理软件可根据这些信息选择一个交换机作为树根. 该方法具体过程为：先计算出每个交换机到通信域组内各进程的最小跳步数，据此找到所有的备选树根. 然后根据备选树根的负载情况对备选树根进行排序，优先使用负载低的树根. 选择好一个树根后，就可以构造一棵物理树，使得每个进程到树根的跳步数最少. 然后检查该物理树的每个交换机是否有空闲的聚合树条目可用. 若每个交换机都有空闲的聚合树条目，则返回该物理树；若失败，则尝试下一个树根.

4.2 构建最小代价Ⅱ型聚合树

由物理树导出的最小代价Ⅱ型聚合树问题. 给定互连网络 $I=(N, L)$ 上一棵关于 M 的物理树 $T=(N_T, L_T)$ 及度约束函数 K，在由物理树 T 导出的Ⅱ型聚合树中求代价最小的聚合树 A^*.

本文利用算法5所示的函数创建最小代价Ⅱ型聚合树. 可以证明算法5产生的聚合树 A^* 是由物理树 T 导出的最小代价Ⅱ型聚合树. 限于篇幅，证明不再赘述.

算法 5. *build_minimum_type_Ⅱ_tree*.

输入：物理树 $T=(N_T, L_T)$，度约束函数 K;

输出：聚合树 $A^*=(N_A, L_A)$.

①记 T 的根节点为 r，若 r 不是交换机，则返回仅包含节点 r 的聚合树；

②否则，对 r 的每个子节点 s_i，记以 s_i 为根的子树为 T_i，调用 *build_minimum_type_Ⅱ_tree* (T_i, K) 产生由物理树 T_i 导出的最小代价Ⅱ型聚合树 A_i^*;

③若 r 仅有1个子节点 T_1，则返回该子节点对应的最小代价Ⅱ型聚合树 A_1^*;

④否则，构造一棵树 $A^*=(N_A, L_A)$，r 为根节点，各 A_i^* 为子树；

⑤若 A^* 中 r 的子节点数小于 r 的度约束 $K(r)$，则

⑥ 从所有 A_i^* 中找到一个最小的叶交换机 L(叶交换机是指所有子节点均是网卡的交换机，最小是指子节点数最小)；

⑦ 从 L 中任选一进程 p，使 p 成为 r 的儿子；

⑧ 若 L 删掉 p 后仅剩1个儿子 q，则将 q 设为 q 祖父的儿子并删掉 L;

⑨ 重复过程⑥~⑧，直到 r 的子节点数等于 $K(r)$ 为止；

⑩返回 A^*.

图11(a)显示了一棵物理树，该树中共有16个进程，使用了13个交换机，每个交换机的度约束均为5；其导出的最小代价Ⅱ型聚合树如图11(b)所示，仅使用了4个交换机.

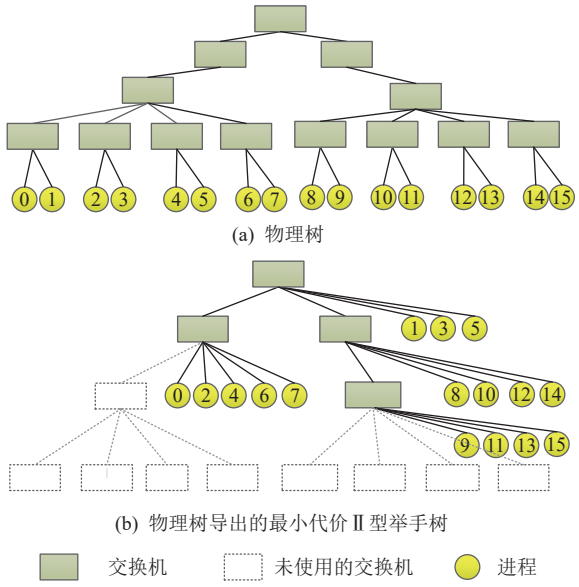


Fig. 11 A physical tree and its reduced minimum cost type Ⅱ aggregate tree

图 11 一棵物理树及其导出的最小代价Ⅱ型聚合树

5 在 2 种聚合树中进行选择

在性能方面, Ⅰ型聚合树与Ⅱ型聚合树各有优势. 对长度较小的集合操作, 聚合包的传输时间占主导, 而Ⅱ型聚合树中聚合包传输路径较短, 故延迟比Ⅰ型聚合树低. 对长度较大的集合操作, 规约计算时间占主导, Ⅰ型聚合树可以将规约计算分布到大量网卡中并发进行, 故延迟较小. 创建通信域时, 需要综合考虑各方面因素, 选择使用Ⅰ型聚合树还是Ⅱ型聚合树.

为简化分析, 假设Ⅱ型聚合树是标准的 d 叉树, 记 $\delta\times l=\alpha$, $\theta+\dfrac{S'}{\beta}+S^*\times\gamma=b$, 则Ⅰ型、Ⅱ型聚合树的时间开销分别如式(7)(8)所示. 为简化讨论, 仅考虑 $S'=S^*=S$ 的情况.

$$T_{\text{aggregate_I}}\approx(a+kb)\log_k P, \tag{7}$$

$$T_{\text{aggregate_II}}\approx\frac{a}{2}+db\log_d P. \tag{8}$$

定义如式(9)所示的函数 $F(d,k,P,S)$:

$$F(d,k,P,S)=T_{\text{aggregate_I}}-T_{\text{aggregate_II}}=(\log_k P-0.5)a+(k\log_k P-d\log_d P)b, \tag{9}$$

$$F'_P=\frac{1}{P}\left(\frac{a+kb}{\ln k}-\frac{db}{\ln d}\right), \tag{10}$$

$$F'_S=\left(\frac{1}{\beta}+\gamma\right)(k\log_k P-d\log_d P). \tag{11}$$

可以根据式(9)~(11)分析 $F(d,k,P,S)$ 的变化趋势, 并据此构造一张静态映射表. 给定集合操作类型及消息长度后, 通过查表即可快速确定聚合树类型及宽度. 表 3 列出了本文实验环境中如何根据聚合包长度来选择使用Ⅰ型聚合树还是Ⅱ型聚合树. 当申请不到创建Ⅱ型聚合树所需的通信资源时, 需要回退到Ⅰ型聚合树.

Table 3 Method to Select Aggregate Tree

表 3 聚合树选择方法

d	聚合包长度/B	聚合树类型	k
16	(0, 2 048]	Ⅱ型	16
64	[0, 323]	Ⅱ型	64
	(323, 512]	Ⅰ型	16
	(512, 2 048]	Ⅰ型	8
不能创建Ⅱ型聚合树时	[0, 64]	Ⅰ型	32
	(64, 512]	Ⅰ型	16
	(512, 2 048]	Ⅰ型	8

6 性能评估

本节将在新一代神威超级计算机中对聚合树性能进行测试.

6.1 Ⅰ型聚合树性能评估

首先测试不同的Ⅰ型聚合树集合消息的延迟. 对同步、广播、8B 长度的规约操作而言, 选择 $k=32$ 可以获得更好的性能, 故分别构造了 32 叉树、32 项树、分层 32 项树. 使用 1 个超节点进行测试, 每个节点运行 2 个进程, 每个进程使用 1 个网络端口, 这 2 个端口间的通信不需要经过网络链路. 分别在 2 种场景下进行测试.

6.1.1 2 层标准胖树中的性能

在 2 层标准胖树中进行测试. 分层 32 项树使用了 3 个层次, 首先将位于同一节点的 2 个进程组成 1 个组, 这 2 个进程间的通信不需要经过网络链路; 然后将位于同一交换机内的 16 个进程组成 1 个组, 该组内的进程通信时仅需经过 2 条链路; 最后将位于同一超节点内的所有进程组成 1 个组, 该组内的进程通信时需经过 4 条链路.

图 12 显示了 Bcast-8B, Bcast-2KB 这 2 种集合操作的测试结果. 可以看出: 1)大部分情况下 32 叉树、32 项树几乎具有相同的延迟; 2)大部分情况下, 分层 32 项树的延迟低 32 叉树、32 项树约 0.6 μs. 分层 32 项树延迟低的原因是: 它先把位于同一节点内的 2 个进程组成 1 个组, 而这 2 个进程间通信不需要经

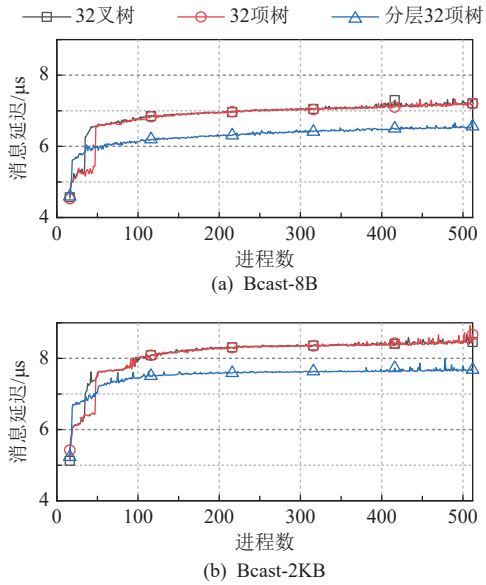


Fig. 12 Messages latency of three types of aggregate trees for different count of processes in standard fat-tree topology

图 12 标准胖树拓扑中不同进程数下 3 种聚合树的消息延迟

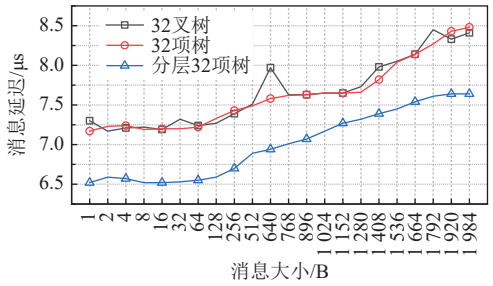


Fig. 13 Messages latency of three types of aggregate trees for different message sizes in standard fat-tree topology

图 13 标准胖树拓扑中不同消息长度下 3 种聚合树的消息延迟

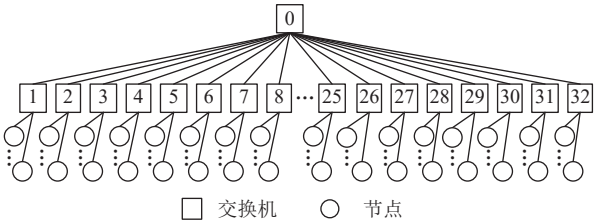


Fig. 14 Illustration of two-layer pruned fat-tree used in the test

图 14 测试使用的 2 层裁剪胖树示意图

过网络链路，故延迟较低. 需要指出的是，进程数较少时分层 32 项树的延迟反而高于另外 2 种聚合树，原因是此时的分层 32 项树的高度大于 32 叉树、32 项树；随着进程数的增多，这 3 种聚合树将具有相同的树高.

不同消息长度时 3 种构造方法下的聚合树的性能如图 13 所示. 测试时使用了 512 个进程，每个进程使用 1 个网络端口. 测试结果与图 12 的测试结果一致，即 32 叉树和 32 项树的性能相当，而分层 32 项树的延迟明显低于 32 叉树和 32 项树.

6.1.2 2 层裁剪胖树中有噪声干扰时的性能

使用如图 14 所示的裁剪胖树进行测试. 该拓扑中，同一交换机内的不同网络端口向其他交换机内的网络端口发消息时会共享一条通信链路. 测试时还在网络中加入了网络噪声，该噪声消耗约 80% 的网络带宽.

分别测试不同进程数时 3 种方法构建的聚合树的性能，并将之与标准胖树下的测试结果进行了对比，结果如图 15 所示. 可以看出，加入网络噪声后，3 种构建方法的集合消息延迟相比标准胖树下有显著增大. 其中 32 叉树增幅最大，分层 32 项树增幅最小.

图 16 显示了进程数为 512 时，3 种方法构建的聚合树中跨组聚合包的数量，其中每条实线上的数字表示该实线所连接的 2 个交换机间的聚合包个数. 在 32 叉树中，大量聚合包发送到 1 号交换机的不同进程内，故每个集合操作中，0 号交换机与 1 号交换机间链路上需传输 496 个聚合包，从而产生严重的链路竞争.

表 4 显示了不同进程数时 3 种聚合树下 Bcast-8 B 的延迟对比，可以看出，在存在噪声干扰的情况下，分层 32 项树的延迟相比传统的聚合树构建方法

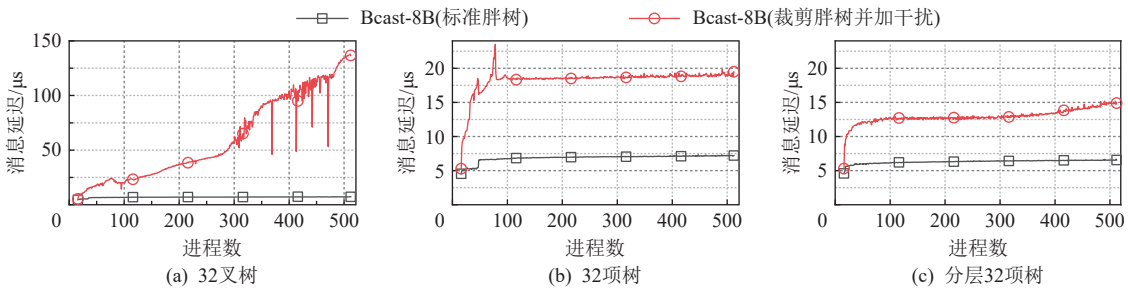
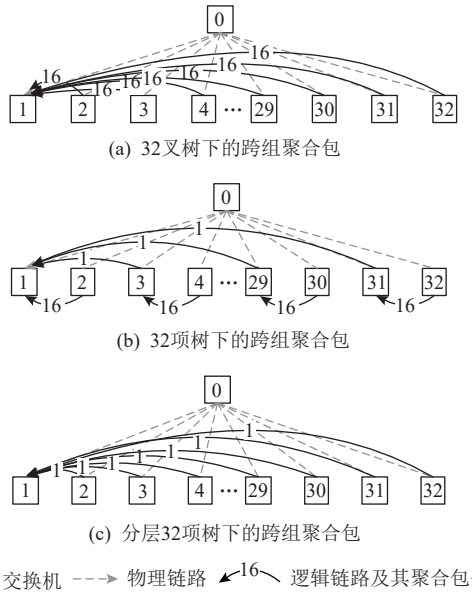


Fig. 15 Collective messages latency in pruned fat-tree topology with inference

图 15 裁剪胖树拓扑下有干扰时集合通信消息延迟



(a) 32叉树下的跨组聚合包

(b) 32项树下的跨组聚合包

(c) 分层32项树下的跨组聚合包

Fig. 16 Inter-group aggregation packets of three types of aggregate trees constructed methods in pruned fat-tree

图 16 裁剪胖树下 3 种聚合树构建方法的跨组聚合包

Table 4 Bcast-8B Latency for Different Count of Processes in Pruned Fat-tree Topology with Inference

表 4 裁剪胖树拓扑下有干扰时不同进程数下 Bcast-8B 延迟

进程数	32 叉树下延迟/μs	32 项树下延迟/μs	分层 32 项树下延迟/μs
64	19.92（↓39%）	17.79（↓32%）	12.11
128	25.11（↓49%）	18.58（↓31%）	12.75
192	35.75（↓64%）	18.54（↓31%）	12.73
256	42.87（↓70%）	18.62（↓32%）	12.75
320	71.81（↓82%）	18.86（↓32%）	12.83
384	97.33（↓86%）	18.94（↓30%）	13.33
448	115.82（↓87%）	18.80（↓26%）	13.94
512	136.60（↓89%）	19.48（↓24%）	14.89

注：括号内的百分比表示分层 32 项树相比 32 叉树、32 项树的延迟下降比例；↓表示分层 32 项树相比 32 叉树、32 项树的延迟下降比例.

下降了 24%~89%. 这表明，相比另外 2 种聚合树构建方法，分层 32 项树具有更少的跨组聚合包，因而存在带宽竞争的情况下可以显著降低集合消息的延迟.

需要注意的是，随着进程数的增大，32 项树的延迟增加较缓慢，而分层 32 项树的延迟增加略快（分层 32 项树相比 32 项树的延迟下降比例由进程数为 64 时的 32% 下降为进程数为 512 时的 24%）. 这是因为，在裁剪胖树中，随着进程数的增大，32 项树中每条链路上传输的最大聚合包数保持为 16，故 32 项树的延迟变化不大；而分层 32 项树中每条链路上传输的最大聚合包数由 1 逐渐增大到 31，故分层 32 项树的延迟会缓慢增加. 但可以肯定的是，随着进程数的进一步增大，分层 32 项树的延迟仍低于 32 项树，这是因为 32 项树中每 2 个相邻交换机间都需要传输 16 个聚合包，从而更易受网络噪声的干扰.

除此之外，本文还在其他场景下测试了 3 种聚合树构建算法的消息延迟. 例如，利用更多超节点进行了测试，其网络拓扑采用 3 层裁剪胖树，故分层 32 项树可采用更多的层次. 此实验的测试结论与上述测试结论一致，即分层 32 项树延迟最低，32 项树次之，32 叉树延迟最高. 限于篇幅，不再赘述其详细测试数据.

6.2 Ⅱ型聚合树性能评估

6.2.1 Ⅱ型聚合树延迟测试

本节对Ⅰ型聚合树及Ⅱ型聚合树的性能进行对比. 测试环境如图 17 所示. 其中Ⅰ型聚合树采用 64 叉树进行构建，Ⅱ型聚合树的树根建在最高层的那个交换机上. 分别测试了不同进程数下 2 种类型聚合树的性能，Bcast-8 B 及 Bcast-2 KB 的延迟如图 18 所示. 可以看出，Ⅱ型聚合树的延迟比Ⅰ型聚合树低

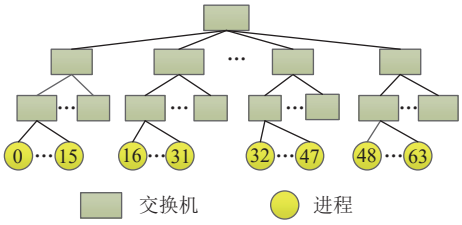


Fig. 17 Network topology used to compare the performance of type Ⅰ and type Ⅱ aggregate trees

图 17 使用网络拓扑对比Ⅰ型聚合树及Ⅱ型聚合树性能

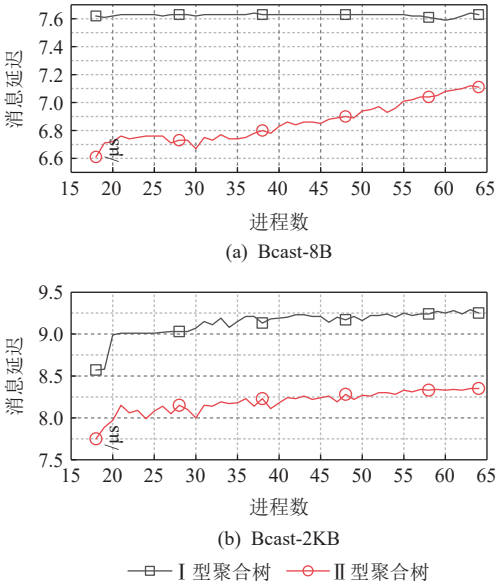


Fig. 18 Performance comparison of type Ⅰ and type Ⅱ aggregate trees

图 18 Ⅰ型聚合树及Ⅱ型聚合树的性能对比

1 μs 左右.

6.2.2 Ⅱ型聚合树资源使用量测试

本节测试最小代价Ⅱ型聚合树构造算法的有效性以及不同通信模式下聚合树条目的使用情况.

MPI 集合通信中，一般将进程划分为 2 维或 3 维网格通信域，其每个维度中的每行或每列都是一个通信域. 本文测试了 5 种典型的通信模式，包括 2 种 2 维网格结构、2 种 3 维网格结构，以及 1 种随机模式. 选择的 2 维网格结构中，256×160 通信模式按拓扑结构划分通信域，模拟拓扑感知的通信模式，即第 1 维有 160 个通信域，故每个通信域都位于同一个超节点内；而 202×202 通信模式存在很多跨超节点的通信域，模拟拓扑无感的通信模式. 2 种 3 维网格结构也是按类似原则进行划分的. 随机模式是指进程随机加入一个通信域.

下面从 2 个方面证明最小代价Ⅱ型聚合树构造算法的有效性：一方面，最小代价Ⅱ型聚合树占用的聚合树条目数相比传统方法构建的Ⅱ型聚合树有显著下降；另一方面，最小代价Ⅱ型聚合树构建算法创建聚合树时的失败率相比传统的Ⅱ型聚合树也有显著下降.

1）聚合树占用的总条目数

首先测试每种通信模式下聚合树占用的总条目数. 假设每个交换机内的聚合树条目数足够多. 对于每种通信模式，依次为每个通信域构建Ⅱ型聚合树，并统计占用的聚合树条目总数，结果如图 19 所示. 其中，传统方法构建的Ⅱ型聚合树是指利用集中式方法构建的物理树. 可以看出，所有通信模式下，最小代价Ⅱ型聚合树占用的条目总数明显低于传统方法构建的聚合树所占用的条目数，最少下降了 90%. 这表明最小代价Ⅱ型聚合树算法在降低聚合条目使用量方面具有显著效果. 有些通信模式下，最小代价Ⅱ型聚合树的聚合树条目总数等于通信域个数，这是因为在这些通信模式下，每个通信域的最小代价Ⅱ型聚合树都仅使用 1 个交换机.

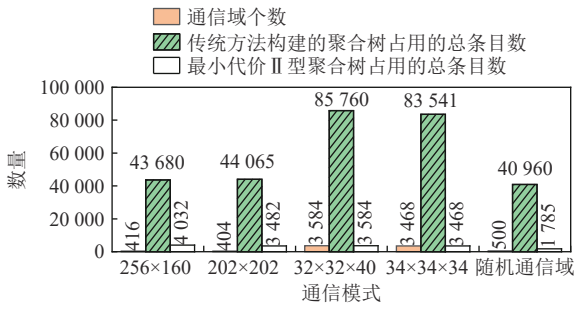


Fig. 19 Total count of aggregate tree entries used by type Ⅱ aggregate trees for different kinds of communication patterns

图 19 不同通信模式下Ⅱ型聚合树使用的总聚合树条目数

2）创建聚合树时的失败率

将每个交换机的聚合树条目数设为 16，然后在每种通信模式下依次为每个通信域构建Ⅱ型聚合树，并记录创建失败的次数. 定义创建聚合树时的失败率为：失败率 = $\frac{\text{失败次数}}{\text{通信域总数}}$，结果如图 20 所示. 可以看出，最小代价Ⅱ型聚合树算法均能为每个通信域成功构建聚合树；而传统的Ⅱ型聚合树构建方法失败率很高，最高可达 74.51%.

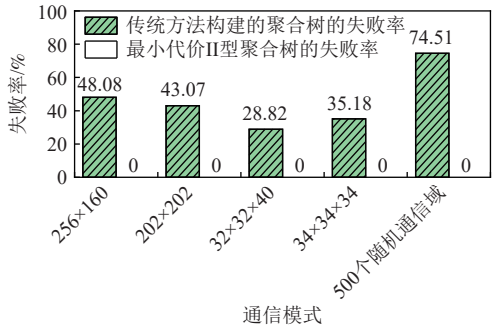


Fig. 20 Failure rate of creating type Ⅱ aggregate trees in different communication patterns

图 20 不同通信模式下创建Ⅱ型聚合树的失败率

需要指出的是，将交换机的聚合树条目数设为更小的值后，某些通信模式下也存在不能创建最小代价Ⅱ型聚合树的情况，但其失败率仍低于传统的Ⅱ型聚合树构建方法，结果不再赘述.

综上所述，最小代价Ⅱ型聚合树算法可以显著减少对交换机内聚合树条目资源的占用，从而可以支持创建更多的Ⅱ型聚合树.

7 结　　论

硬件集合通信中，聚合树对集合操作的性能具有重要影响. 构建聚合树时，需要综合考虑聚合树半径、宽度、负载均衡、网络噪声等的影响. 本文提出了硬件集合通信开销模型，并据此提出了 3 种构建聚合树的方法，包括：1）根据聚合包大小确定Ⅰ型聚合树的宽度；2）最小高度分层Ⅰ型聚合树构建方法；3）最小代价Ⅱ型聚合树构建方法. 然后在神威互连网络中对聚合树构建方法进行了全面测试，在存在网络噪声的情况下，分层Ⅰ型聚合树的消息延迟相比传统构建方法下降了 24%~89%；最小代价Ⅱ型聚

合树使用的交换机聚合条目数相比传统构建方法下降了约90%.

下一步,我们将利用真实的应用程序对聚合树构建方法的性能进行更全面的测试.另外,我们还将研究如何利用硬件集合通信提升长度较大的规约、广播等集合操作的性能,扩大硬件集合通信的适用范围.

作者贡献声明: 陈淑平提出研究思路,设计算法、实验,分析实验数据并撰写论文;尉红梅指导算法及完善实验方案;王飞、李祎负责算法实现、测试数据的整理与分析;何王全、漆锋滨提出研究课题,把握论文创新性,并对论文进行修改.

参 考 文 献

[1] Liao Xiangke, Lu Kai, Yang Canqun, et al. Moving from exascale to zettascale computing: Challenges and techniques[J]. Frontiers of Information Technology & Electronic Engineering, 2018, 19(10): 1236−1244

[2] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard, version 4.0 [S/OL]. (2021-06-09)[2023-02-25]. https://www.mpi-forum.org/docs/mpi-4.0/mpi40-report.pdf

[3] Thakur R, Rabenseifner R, Gropp W. Optimization of collective communication operations in MPICH[J]. International Journal of High Performance Computing Applications, 2005, 19(1): 49−66

[4] Chan E, Heimlich M, Purkayastha A, et al. Collective communication: Theory, practice, and experience[J]. Concurrency and Computation: Practice and Experience, 2007, 19(13): 1749−1783

[5] Richard L G, Devendar B, Lui Pak, et al. Scalable hierarchical aggregation protocol (SHARP): A hardware architecture for efficient data reduction [C/OL] //Proc of the 1st Int Workshop on Communication Optimizations in HPC. Piscataway, NJ: IEEE, 2016 [2023-02-25]. https://doi.org/10.1109/COMHPC.2016.006

[6] Sapio A, Abdelaziz I, Aldilaijan A, et al. In-network computation is a dumb idea whose time has come [C] //Proc of the 16th ACM Workshop on Hot Topics in Networks. New York: ACM, 2017: 150−156

[7] Benson T A. In-network compute: Considered armed and dangerous [C] //Proc of the 17th Workshop on Hot Topics in Operating Systems. New York: ACM, 2019: 216−224

[8] Chen Dong, Eisley N A, Heidelberger P, et al. The IBM Blue Gene/Q interconnection fabric[J]. IEEE Micro, 2012, 32(1): 32−43

[9] Chen Dong, Eisley N A, Heidelberger P, et al. The IBM Blue Gene/Q interconnection network and message unit [C/OL] //Proc of the 24th Int Conf for High Performance Computing, Networking, Storage, and Analysis. Piscataway, NJ: IEEE, 2011 [2023-02-25]. https://doi.org/10.1145/2063384.2063419

[10] Chen Dong, Eisley N, Heidelberger P, et al. Looking under the hood of the IBM blue gene/Q network [C/OL] //Proc of the 25th Int Conf for High Performance Computing, Networking, Storage, and Analysis. Piscataway, NJ: IEEE, 2012 [2023-02-25]. https://doi.org/10.1109/SC.2012.72

[11] Kumar S, Mamidala A, Heidelberger P, et al. Optimization of MPI collective operations on the IBM Blue Gene/Q supercomputer[J]. The International Journal of High Performance Computing Applications, 2014, 28(4): 450−464

[12] Manjunath G V, Gilad S, Richard L G, et al. Accelerating OpenSHMEM collectives using in-network computing approach [C]// Proc of the 31st Int Symp on Computer Architecture and High Performance Computing. Piscataway, NJ: IEEE, 2019: 212−219

[13] Mellanox Technologies Ltd. Aggregation Protocol: US, US10284383 [P]. 2019-05-07

[14] Bharath R, Kaushik K S, Nick S, et al. Scalable MPI collectives using SHARP: Large scale performance evaluation on the TACC Frontera system [C] //Proc of the 1st Workshop on Exascale MPI. Piscataway, NJ: IEEE, 2020: 11−20

[15] Gao Jiangang, Lu Hongsheng, He Wangquan, et al. The interconnection network and message machinasim of Sunway exascale prototype system[J]. Chinese Journal of Computers, 2021, 44(1): 222−234 (in Chinese)
(高剑刚, 卢宏生, 何王全, 等. 神威E级原型机互连网络和消息机制[J]. 计算机学报, 2021, 44(1): 222−234)

[16] Zimmer C, Atchley S, Pankajakshan R, et al. An evaluation of the CORAL interconnects [C/OL] //Proc of the 32nd Int Conf for High Performance Computing, Networking, Storage, and Analysis. Piscataway, NJ: IEEE, 2019[2023-02-25]. https://doi.org/10.1145/3295500.3356166

[17] Hu Meiyong. MPI collective communication optimization on Tianhe high-speed interconnect[D]. Changsha: National University of Defense Technology, 2014 (in Chinese)
(胡美勇. 基于"天河"高速互连网络的MPI聚合通信优化[D]. 长沙: 国防科学技术大学, 2014)

[18] Rottenstreich O, Yallouz J, Levi L. Isolated trees in multi-tenant fat tree datacenters for in-network computing [C] //Proc of the 27th IEEE Symp on High-Performance Interconnects. Piscataway, NJ: IEEE, 2020: 55−62

[19] Banerjee S, Kommareddy C, Kar K, et al. Construction of an efficient overlay multicast infrastructure for real-time applications [C] //Proc of the 22nd IEEE INFOCOM. Piscataway, NJ: IEEE, 2003: 1521−1531

[20] Ho J M, Lee D T, Chang C H, et al. Minimum diameter spanning trees and related problems[J]. SIAM Journal on Computing, 1991, 20(5): 987−997

[21] Shi S Y, Turner J S, Waldvogel M. Dimensioning server access bandwidth and multicast routing in overlay networks [C] //Proc of the 10th IEEE Int Workshop on Network and Operating System Support for Digital Audio and Video. Piscataway, NJ: IEEE, 2001: 83−91

[22] Valerio M, Moser L E, Melliar-Smith P M. Recursively scalable fat-

trees as interconnection networks［C］// Proc of the 13th IEEE Annual Int Phoenix Conf on Computers and Communications. Piscataway, NJ: IEEE, 1994: 40−46

［23］ Petrini F, Vanneschi M. k-ary n-trees: High performance networks for massively parallel architectures［C］//Proc of the 11th Int Parallel Processing Symp. Piscataway, NJ: IEEE, 1997: 87−93

［24］ Kim J, Dally W J, Scott S, et al. Technology-driven, highly-scalable dragonfly topology［C］//Proc of the 35th Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2008: 77−88

［25］ Zhao Tianhai, Wang Yunlan, Wang Xu. Optimized reduce communication performance with the tree topology［C］//Proc of the 4th High Performance Computing and Cluster Technologies Conf. New York: ACM, 2020: 165−171

［26］ Tipparaju V, Nieplocha J, Panda D. Fast collective operations using shared and remote memory access protocols on clusters［C/OL］//Proc of the 17th Int Parallel and Distributed Processing Symp. Piscataway, NJ: IEEE, 2003［2023-02-25］. https://doi.org/10.1109/PDPS.I2003.1213188

［27］ Jain S, Kaleem R, Balmana M G, et al. Framework for scalable intra-node collective operations using shared memory［C/OL］//Proc of the 31st ACM/IEEE Int Conf for High Performance Computing, Networking, Storage and Analysis. Piscataway, NJ: IEEE, 2018［2023-02-25］. https://doi.org/10.1109/SC.2018.00032

［28］ Luo Xi, Wu Wei, Bosilca G, et al. HAN: A hierarchical autotuned collective communication framework［C］//Proc of the 22nd Cluster Conf. Piscataway, NJ: IEEE, 2020: 23−34

［29］ Li Shigang, Hoefler T, Snir M. Numa-aware shared-memory collective communication for MPI［C］//Proc of the 22nd Int Symp on High-Performance Parallel and Distributed Computing. New York: ACM, 2013: 85−96



Chen Shuping, born in 1983. PhD, associate professor. His main research interests include high performance computing and interconnect networks.

陈淑平, 1983 年生. 博士, 副研究员. 主要研究方向为高性能计算、高速互连网络.



Wei Hongmei, born in 1968. professor, PhD supervisor. Member of CCF. Her research interests include parallel computing and parallel compilation.

尉红梅, 1968 年生. 研究员, 博士生导师. CCF 会员. 主要研究方向为并行计算、并行编译.



Wang Fei, born in 1981. Associate professor. His research interests include compiler design and optimization, and artificial intelligence.

王　飞, 1981 年生. 副研究员. 主要研究方向为编译器设计及优化、人工智能.



Li Yi, born in 1994. Master, engineer. Her main research interests include high performance computing and high-speed interconnect networks.

李　祎, 1994 年生. 硕士, 工程师. 主要研究方向为高性能计算、高速互连网络.



He Wangquan, born in 1975. PhD, professor. His research interests include parallel programming language design and compiler optimization.

何王全, 1975 年生. 博士, 研究员. 主要研究方向为并行计算编程语言设计、编译器优化.



Qi Fengbin, born in 1966. Professor, PhD supervisor. Fellow of CCF. His research interests include high performance computing architecture, compiler optimization, and parallel algorithm.

漆锋滨, 1966 年生. 正高级工程师, 博士生导师, CCF 会士. 主要研究方向为高性能计算体系结构、编译器优化、并行算法.