软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(10):2509−2522 [doi: 10.3724/SP.J.1001.2011.03915]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

非平衡进程到达模式下 MPI 广播的性能优化方法*

刘志强+, 宋君强, 卢风顺, 徐 芬

(国防科学技术大学 计算机学院,湖南 长沙 410073)

Optimizing Method for Improving the Performance of MPI Broadcast under Unbalanced Process Arrival Patterns

LIU Zhi-Qiang+, SONG Jun-Qiang, LU Feng-Shun, XU Fen

(College of Computer, National University of Defense Technology, Changsha 410073, China)

+ Corresponding author: E-mail: liuzq@nudt.edu.cn

Liu ZQ, Song JQ, Lu FS, Xu F. Optimizing method for improving the performance of MPI broadcast under unbalanced process arrival patterns. *Journal of Software*, 2011,22(10):2509−2522. http://www.jos.org.cn/1000-9825/3915.htm

Abstract: This paper aims at improving the performance of MPI broadcasts under unbalanced process arrival (UPA) patterns. This paper analyzes this problem with a performance model and proves that the negative impact of UPA on MPI broadcast can be effectively reduced by the competition of intra-node MPI processes on a multicore cluster. Based on this theory, a new optimizing method, called competitive and pipelined method (CP), is proposed. The CP method can start inter-node communications during the broadcast process through an intra-node competitive mechanism. In a CP method based broadcast algorithm, intra-node communications overlap inter-node communications through a pipelined method, and intra-node communications are implemented through shared memory while inter-node communications are executed by a set of leader MPI processes, which is selected by the competitive mechanism. In order to verify the CP method, this paper improves three typical broadcast algorithms by using this method and evaluates these algorithms in a real platform by using a micro-benchmark case and two practical applications. The results show that the performance of the CP method can effectively improve the performance of broadcast algorithms in the condition of UPA patterns. In the experimental results of the performance of the practical applications, the performance of CP broadcasts is about 16% higher than the performance of P broadcasts and is 18% to 24% higher than the performance of broadcast operation in MVAPICH2 1.2.

Key words: process arrival pattern; MPI; collective communication; MPI_Bcast; competitive and pipelined method

摘 要: 为了提高非平衡进程到达(unbalanced process arrival,简称 UPA)模式下 MPI 广播的性能,对 UPA 模式下的广播问题进行了理论分析,证明了在多核集群环境中通过节点内多个 MPI 进程的竞争可以有效减少 UPA 对 MPI 广播性能的影响,并在此基础上提出了一种新的优化方法,即竞争式流水化方法(competitive and pipelined method,简

* 基金项目: 国家自然科学创新群体基金(60621003)
收稿时间: 2009-12-11; 修改时间: 2010-03-05; 定稿时间: 2010-06-28

称 CP).CP 方法通过一种节点内进程竞争机制在广播过程中尽早启动节点间通信,经该方法优化的广播算法利用共享内存在节点内通信,利用由竞争机制产生的引导进程执行原算法在节点间通信.并且,该方法使节点间通信和节点内通信以流水方式重叠执行,能够有效利用集群系统各节点的多核优势,减少了 MPI 广播受 UPA 的影响,提高了性能.为了验证 CP 方法的有效性,基于此方法优化了 3 种典型的 MPI 广播算法,分别适用于不同消息长度的广播.在真实系统中,通过微基准测试和两个实际的应用程序对 CP 广播进行了性能评价,结果表明,该方法能够有效地提高传统广播算法在 UPA 模式下的性能.在应用程序的负载测试实验结果中,CP 广播的性能较流水化广播的性能提高约 16%,较 MVAPICH2 1.2 中广播的性能提高 18%~24%.

关键词: 进程到达模式;MPI;集合通信;MPI_Bcast;竞争式流水化方法

中图法分类号: TP316 **文献标识码:** A

MPI 是目前高性能计算领域并行计算程序开发的事实标准.它以支持消息传递通信为主要目的,除了提供基本的点对点通信接口以外,MPI 还抽象了一些常用的全局通信模式作为集合通信(collective communication)接口提供给用户.大多基于 MPI 开发的并行计算程序会利用集合通信作为并行算法的通信核心,从而集合通信的性能成为决定程序整体性能的一大关键因素.随着应用程序并行规模的扩大,集合通信对整体程序的性能有更加重要的影响.因此,改善和优化集合通信性能的意义十分重大,也是一个非常活跃的研究课题.

在优化集合通信性能的研究过程中,多数工作关注于面向具体的体系结构和网络拓扑提出新的通信算法.其共同特点是:基于消息长度和通信域的静态特征进行优化,通过有效利用信道带宽或减少通信次数来减少通信的整体耗时;同时,它们在设计、实现高效集合通信算法时,潜在的共同假设是通信域中的进程同时到达执行集合通信.然而最近的一些研究[1]发现,在 MPI 并行程序运行时,各个 MPI 进程到达执行同一次集合通信的时刻往往差别较大,这一现象对集合通信的性能有非常显著的影响.文献[1]将各个进程到达执行同一次集合通信的时刻定义为进程到达模式(process arrival pattern).本文扩展了这一定义,将各个 MPI 进程的到达时刻存在明显差异的进程到达模式定义为非平衡进程到达(unbalanced process arrival,简称 UPA)模式,反之为平衡进程到达(balanced process arrival,简称 BPA)模式.UPA 是 MPI 并行任务运行时的常态,提高集合通信性能在 UPA 模式下的性能对提高集合通信的实际使用性能有着十分重要的意义.

广播操作是最常用的集合通信操作之一,该操作可以将消息从一个进程(称为根进程)发送到通信域中其他所有进程,MPI 标准将广播操作定义为 MPI_Bcast[2].目前,主要的高效广播算法包括适用于短消息的二项树(binomial tree)算法[3]、适用于长消息的先散发后组收集的算法[4]和适用于多层次系统的流水化广播算法[5−7].这些算法被广泛应用在目前的主流 MPI 库中,比如 MPICH2[4],Open MPI[8]和 NEC MPI[9].在 BPA 模式下,这些算法都能够进行高效广播;但在 UPA 模式下,这些算法的性能也都会受 UPA 较大的影响,从而使性能有所降低[10].

本文针对 UPA 模式下 MPI 广播的性能优化问题,提出了一种新的竞争式流水化(competitive and pipelined,简称 CP)方法.该方法适用于目前被高性能计算领域广泛使用的多核(多处理器)集群,并且有效利用了此类系统节点内包含多个核心(处理器)的特点.经该方法优化的广播算法利用共享内存在节点内通信,并且利用由竞争产生的引导进程执行原算法在节点间进行通信;同时,通过该方法能够使得节点间通信和节点内通信以流水方式重叠执行.经 CP 方法优化的广播算法与传统流水化广播算法的区别在于:在传统流水化算法中,引导进程由预先确定的进程担任;但在 CP 广播算法中,引导进程由一种竞争机制动态产生.这种竞争机制是减少算法受非平衡进程到达影响的关键.

本文从理论上分析了 UPA 模式对广播算法造成的影响,证明了在多核集群环境中通过节点内多个 MPI 进程的竞争可以有效减少 UPA 对 MPI 广播性能的影响,并在此基础上详细阐述了 CP 方法的设计原理.为了验证方法的有效性,本文利用 CP 方法优化了 3 种典型的广播算法,其中包括二项树算法(binomial tree,简称 BT)、先分散后递归倍增组收集广播算法(recursive doubling allgather after scatter,简称 RDAAS)和先分散后环组收集广播算法(ring allgather after scatter,简称 RingAAS),并在真实平台上进行了微基准测试(micro benchmark)和实际应用负载测试等性能测试实验.在应用程序的负载测试实验结果中,CP 广播的性能较流水化广播的性能提高约

16%,较 MVAPICH2 1.2 中广播的性能提高 18%~24%.所有实验结果表明,CP 方法可以有效地减少广播算法性能受 UPA 的影响,从而提高广播算法在 UPA 模式下的性能.

本文第 1 节介绍相关工作.第 2 节在对 UPA 问题进行详细分析的基础上提出 CP 方法的设计原理.第 3 节描述基于 CP 方法的广播算法在 MPI 加速器上实现的关键技术.第 4 节介绍在真实平台上进行详细的性能测试实验情况.最后一节对本文的工作进行总结,并提出对未来工作的展望和设想.

1 相关工作

进程到达模式影响是 2007 年由 Faraj 等人提出的概念[1],近年来,进程到达模式对集合通信性能的影响问题迅速受到广泛的关注[10−12].Faraj 等人在文献[12]中介绍了一种 BlueGene BG/P 系统上的异步集合通信,它受 UPA 的影响较小,但需要专门的硬件支持,并不适用于目前大多数的并行系统.文献[11]提出了一种适应于任意进程到达模式的高效 MPI_Alltoall 通信算法,它充分利用了 Infiniband 上的高效 RDMA 和节点内共享内存进行通信.文献[10]提出了一种适应任意进程到达模式的超长消息广播优化方法,它通过发送控制报文获得进程到达状态,据此优化通信过程,但会带来较大的额外开销,因此只适用于超长消息与超不平衡进程到达模式.本文提出的竞争式流水化优化方法适用于目前应用最普遍的多核集群系统,不需要额外的控制报文或产生额外的开销,经其优化的广播算法可以适应任意的进程到达模式,同时不受消息长度的限制.

2 问题分析与竞争式流水化优化方法设计

本节以二项树广播算法(下文称 BT 广播算法)为例定量分析 UPA 对算法性能的影响,并详细介绍一种用于改进算法减少受 UPA 对性能影响的 CP 方法,此方法是本文的主要贡献.

2.1 性能模型

定义 1. 设进程 i 到达执行通信的时刻为 t_i,完成通信的时刻为 f_i,则称 f_i-t_i 为进程 i 的通信耗时,记为 T_i.有 $T_i=W_i+C_i$,其中,W_i 为由于通信上的依赖关系所必须的等待耗时,C_i 为数据传输耗时.对于随机到达进程的通信,在理想情况下,C_i 只受信道性能与消息大小的影响,W_i 是随机的.称 $E(W_i)$为进程 i 的平均等待耗时,并称 $E(T_i)$为进程 i 的平均通信耗时,有 $E(T_i)=E(W_i)+C_i$.

定义 2. 称 $\max_{i=0,\ldots,p-1}(T_i)$为通信耗时,记为 T,其中,p 为参与通信的进程数量.对于随机到达进程的通信,称 $\max_{i=0,\ldots,p-1}(E(T_i))$为平均通信耗时,记为 $\overline{T}$.

在下文的分析中,不失一般性,所研究的广播都以 0 号进程为根进程,另外,我们假设:

- 任意两个进程通过点对点信道传输大小为 m 字节的消息需要$\alpha+m\cdot\beta$单位时间(秒),其中:α表示传输延迟;β表示每字节的传送时间,β与带宽成反比.设函数 $t(m)=\alpha+m\cdot\beta$.
- 在 UPA 模式下,进程到达执行集合通信的时间区间为$[0,k]$,并且服从均匀分布.
- 为便于分析,假设进程间传输数据采用同步通信模式,即需要发送方和接收方都已到达执行发送和接收操作时传输才能进行,并且发送方和接收方同时结束数据传输.这一假设不影响通信耗时的计算.

由 BT 广播算法的定义和本文的假设可知:当 p 个进程利用二项树算法广播大小为 m 字节的消息时,$C_0=\lceil\log_2 p\rceil\cdot t(m)$,下文以根进程为对象研究广播的通信耗时,并省略符号的下标.

BPA 模式下的 BT 广播如图 1 所示,在此情况下,$W\equiv 0$,p 个进程利用二项树算法广播大小为 m 字节消息的通信耗时为

$$T=\lceil\log_2 p\rceil\cdot t(m) \tag{1}$$

UPA 模式下的 BT 广播如图 2 所示,它与实际情况更相符合,等待未就绪进程耗费的时间影响广播的通信耗时,在这种情况下,

$$\overline{T}=E(W)+\lceil\log_2 p\rceil\cdot t(m) \tag{2}$$

其中,$W=\sum_{i=0}^{\lceil\log_2 p\rceil-1}\omega_i$, ω_i 为进行第 i 步通信数据传输前所需要的等待时间.对比公式(1)、公式(2)可知,$E(W)$反映

了进程非平衡到达对通信性能的影响.不同的广播算法在相同的条件下,$E(W)$越大,则说明算法容忍非平衡到达进程的能力越差.为了更清晰地分析算法容忍非平衡到达进程的能力,我们有如下定义.

定义 3. 两进程传递大小为 m 字节的消息,如果进程随机到达执行通信的时间区间为$[0,k]$,称 $k/t(m)$为非平衡度(unbalance degree),记为 Ψ.

定义 4. 若存在函数φ,有

$$\varphi(\Psi)=E(W)/t(m) \tag{3}$$

则称φ为影响模型,其值称为非平衡影响系数(unbalance impact coefficient),记为Γ.

由定义 4 可以得到与非平衡影响系数相关的性能模型:

$$\overline{T}=\Gamma\cdot t(m)+\lceil\log_2 p\rceil\cdot t(m) \tag{4}$$

综上可知,Γ反映了广播受 UPA 影响的程度.本节的以下内容分为两个部分:第 2.2 节详细分析 BT 广播算法的影响模型,第 2.3 节介绍 CP 方法的设计.

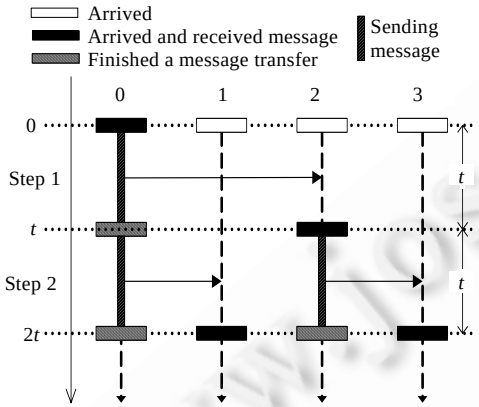

Fig.1 BPA binomial tree broadcast

图 1 平衡进程到达的二项树广播

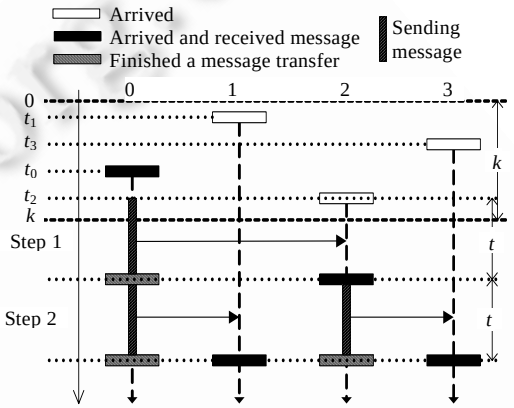

Fig.2 UPA binomial tree broadcast

图 2 非平衡进程到达的二项树广播

2.2 问题分析

在研究 UPA 模式下 BT 广播的性能之前,我们首先研究 UPA 模式下点对点通信的平均通信耗时,这对于研究广播的平均通信耗时具有重要的意义.根据假设,可以得到 UPA 模式下点对点通信的平均耗时,如定理 1 所述.

定理 1. 当两个进程间传递大小为 m 字节的消息时,如果进程到达执行通信的时间区间为$[0,k]$,并且服从此区间上的均匀分布,则通信的平均耗时为 $k/6+t(m)$.

证明:设 p_s 为发送方进程,p_r 为接收方进程,x 与 y 分别为 p_s 与 p_r 到达执行点对点通信时刻的随机变量,且服从$[0,k]$上的均匀分布.令 $t=\alpha+m\cdot\beta$.

p_s 的平均通信耗时为 $E(T_s)=E(W_s)+t(m)$,p_r 的平均通信耗时为 $E(T_r)=E(W_r)+t(m)$.其中:

$$W_s=\begin{cases}y-x, & x<y\\ 0, & x\geqslant y\end{cases},\ W_r=\begin{cases}x-y, & x>y\\ 0, & x\leqslant y\end{cases},$$

则有:

$$E(W_s)=\int_0^k\int_x^k(y-x)\cdot\frac{1}{k^2}\mathrm{d}y\mathrm{d}x=\int_0^k\int_y^k(x-y)\cdot\frac{1}{k^2}\mathrm{d}x\mathrm{d}y=E(W_r)=\frac{k}{6}.$$

可知 $E(T)=k/6+t(m)$.证毕. □

研究 p 个进程参与的 BT 广播,设 $b_i=2^{\lceil\log_2 p\rceil-1-i}$,根进程进行广播时的第 i 个通信步骤可以描述为:等待 b_i 号进程到达后,将消息传输给 b_i 号进程.因此,与研究点对点通信的方法类似,可以得到每步等待时间ω_i:

$$W_i=\begin{cases}t_{b_i}-\sum_{j=0,\ldots,i-1}c_j-t_0, & \sum_{j=0,\ldots,i-1}c_j+t_0<t_{b_i},i\geqslant 1\\ 0, & \sum_{j=0,\ldots,i-1}c_j+t_0\geqslant t_{b_i},i\geqslant 1\\ t_{b_i}-t_0, & t_0<t_{b_i},i=0\\ 0, & t_0\geqslant t_{b_i},i=0\end{cases} \tag{5}$$

其中:$i\in\{0,1,\ldots,\lceil\log_2 p\rceil-1\}$;$c_i=W_i+t(m)$为第 i 个通信步的耗时;t_j 为进程 P_j 到达执行广播的时间随机变量,服从[0,k]上的均匀分布,其中,$j\in\{0,1,\ldots,p-1\}$.

当 i=0 时,由定理 1 可得 $E(\omega_0)=k/6$;

当 $i\geqslant 1$ 时,扩展定理 1,用 $E(c_j)$近似 c_j,可以求得:

$$E(\omega_i)\approx\begin{cases}\dfrac{\left(k-\sum_{j=0,\ldots,i-1}E(c_j)\right)^3}{6k^2}, & \sum_{j=0,\ldots,i-1}E(c_j)<k\\ 0, & \sum_{j=0,\ldots,i-1}E(c_j)\geqslant k\end{cases} \tag{6}$$

将公式(6)代入公式(3),通过数值方法可以求得 BT 广播算法的影响模型,图 3 中标示为“BT broadcast”的曲线表示了这一模型.图 4 中标示为“BT 广播(非平衡)”与“BT 广播(平衡)”的曲线对比了模拟的 BT 广播算法在 BPA 模式和 UPA 模式下的性能,广播的实验参数假设为 p=1024,α=4μs,β=10^{-2}μs/byte,k=500μs,节点数为 128,进程在节点中平均分配.

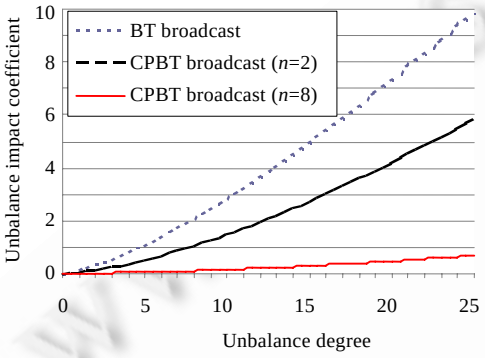

Fig.3 Impact model

图 3 影响模型

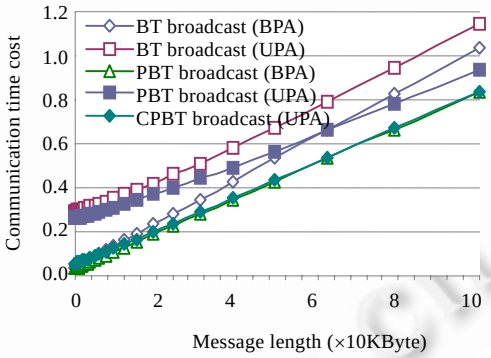

Fig.4 Performance of broadcast algorithm

图 4 广播算法性能

对比可以看出,BT 广播受到 UPA 的影响显著.

2.3 竞争式流水化优化方法

流水化方法优化集合通信的思路是,将通信按节点内通信和节点间通信分为两级,并将两级通信流水化并行,节点间通信采用原有算法进行,节点内充分利用共享内存的优势进行通信,各个节点负责进行节点间通信的进程称为引导进程(leader).

在传统的流水化方法中,每个节点内的引导进程都是被预先确定的,即使它有可能比节点内的其他进程到达的时间都要晚.所以,此方法仅能通过缩短集合通信算法的通信步数有效地减少数据传输耗时(C),但对算法影响模型的改善甚微.

我们把利用传统的流水化方法优化的 BT 广播算法称为流水化 BT 广播算法,如图 5 所示.优化后的算法利用 BT 算法进行节点间广播,利用共享内存进行节点内广播,并且节点内广播在节点间通信的同时进行.流水化 BT 广播的通信耗时等于最后得到消息的引导进程的通信耗时,也等于根进程进行节点间通信的耗时加一次节点内广播的耗时.设节点内广播的耗时为 $r(m)$,在下文中,我们假设 $r(m)$=$t(m)$.

如果参与通信的节点数目为 N,在 BPA 模式下,流水化 BT 广播的通信耗时为$\lceil\log_2 N\rceil t(m)+r(m)$;在 UPA 模式

下,若节点数目足够多,流水化 BT 算法的影响模型与 BT 算法的影响模型(图 3 中标示为“BT 广播”的曲线)相同,即算法性能仍然会受到 UPA 的严重影响.图 4 用模拟的实验结果说明了这一问题,图中标示为“PBT 广播(非平衡)”与“PBT 广播(平衡)”的曲线对比了流水化二项树广播算法在 BPA 模式和 UPA 模式下的性能,实验参数与第 2.2 节中的广播参数相同.

本文提出的 CP 方法改进了传统的流水化方法,使其能够从根本上改善算法的影响模型.这一优点得益于在 CP 方法中引导进程由节点内最早到达的进程担任,其到达时间分布与定理 2 所述相同.

定理 2. 设节点内参与通信的进程数为 n,若进程到达执行通信的时间区间为$[0,k]$,并服从此区间上的均匀分布,则竞争式流水化方法中引导进程到达时间(τ)的分布函数为

$$P(\tau<x)=1-\left(1-\frac{x}{k}\right)^n.$$

证明:设 t_i 为进程 i 到达执行通信的时间随机变量,其中,$i\in\{0,1,\ldots,n-1\}$.根据命题可知,$\tau=\min_{0,\ldots,n-1}t_i$.

$$P(\tau<x)=P(\min_{0,\ldots,n-1}t_i<x)=1-P(\min_{0,\ldots,n-1}t_i>x)=1-\prod_{i=0,\ldots,n-1}P(t_i>x)=1-\left(1-\frac{x}{k}\right)^n.$$ □

图 6 对比了传统流水化方法与 CP 方法中引导进程到达时间的概率分布.从图中可知:通过 CP 方法,算法中引导进程的到达时间分布被显著改善.同时,曲线也显示,节点规模越大,对引导进程的到达时间分布优化越显著.在下文中,我们把由 CP 方法改进的 BT 广播简称为 CPBT 广播.

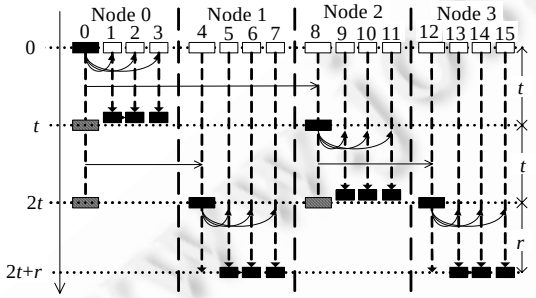

Fig.5 Pipelined binomial tree broadcast algorithm

图 5 流水化二项树广播算法

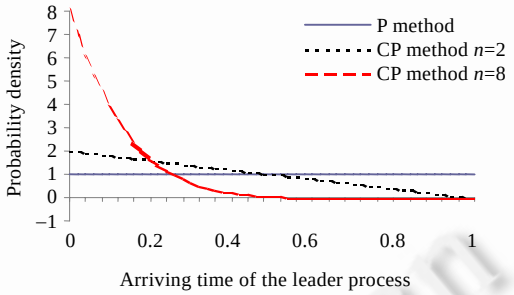

Fig.6 Probability density of the arriving time of the leader process

图 6 引导进程到达时间的概率密度

利用第 2.2 节的分析方法研究 $p=N\times n$ 个进程参与的 CPBT 广播,其中,N 为节点数目,n 为每个节点内参与广播通信的进程数目.我们可以得到根进程进行节点间通信每步的平均等待时间:

$$E(\omega_i)\approx\begin{cases}\dfrac{\left(k-\sum_{j=0,\ldots,i-1}E(c_j)\right)^{n+2}}{(n+1)(n+2)k^{n+1}}, & \sum_{j=0,\ldots,i-1}E(c_j)<k, i\geqslant 1\\ 0, & \sum_{j=0,\ldots,i-1}E(c_j)\geqslant k, i\geqslant 1\\ \dfrac{k}{(n+1)(n+2)}, & i=1\end{cases}\tag{7}$$

将公式(7)代入公式(3),通过数值方法可以求得 CPBT 广播算法的影响模型.图 3 中标示有“CPBT 广播”的两条曲线分别表示了进程为 2 和 8 时的这一模型,它们都比 BT 广播算法和流水化 BT 广播算法的影响模型有明显的改善.同时,曲线也显示出,节点规模越大,CPBT 广播算法的影响模型越优.图 4 中标示为“CPBT 广播(非平衡)”的曲线显示了 CPBT 算法在 UPA 模式下的性能,条件参数与第 2.2 节中的实验参数一致.对比 BPA 模式下流水化 BT 广播算法的性能(图 4 中标示为“PBT 广播(平衡)”的曲线)可以发现,通过 CP 方法优化的 BT 广播算法受 UPA 的影响非常小,其性能明显优于传统的 BT 广播算法和流水化 BT 广播算法.

同理,也可以使用 CP 方法优化适用于长消息的先分散后组收集的广播算法,本文将在下一节中介绍应用 CP 方法实现优化广播算法的一些关键问题.

3 MPI 加速器中的竞争式流水化广播算法实现

通过第 2 节介绍的 CP 方法优化 MPICH2 中常用的广播算法可以得到如图 7 所示的 CP 广播算法.它由适用于广播短消息的 BT 广播算法和适用于广播长消息的先分散后组收集的广播算法(AAS)组成.在此算法中,get_coll_request_ptr 用于获得集合通信请求对象,除根进程所在的节点外,其余节点中最早执行此函数的进程“竞争”成为本次广播通信的引导进程;Inter_BT_bcast,Inter_BT_scatter,Inter_RD_allgather 和 Inter_RING_allgather 分别为节点间广播、分散和组收集例程,其中,BT,RD,RING 分别表示二项树算法、递归倍增算法和环算法,这些例程接口的最后一个参数是当例程接收到新消息就立即调用的节点内广播回调函数.PDMPIC_intra_bcast 和 PDMPIC_intra_bcastseg 分别表示节点内广播和节点内消息分段广播,对于引导进程,这两个操作都是非阻塞的.即引导进程将消息或消息分段登记到通信请求对象中后立即返回,而不需要等节点中所有的相关进程都得到数据后才返回;对于非引导进程,这两个操作是阻塞的.即接收到新消息或新消息分段后才返回,算法根据 PDMPIC_intra_bcastseg 的返回值判断本 MPI 进程是否已接收完整个消息.通过以上机制,节点间通信与节点内通信得以流水化重叠进行.理论上,这是一种高效的广播算法.

```
1  Algorithm. MPI_Bcast
   Input:buf, count, datatype, root, comm;
2  MPI_Type_size(datatype,&typesize);
3  nbyte=typesize*count;
4  MPI_Comm_rank(&rank,MPI_COMM_WORLD);
5  get_coll_request_ptr(&request,PDMPI_REQ_BCAST,comm);
6  if nbyte<BCAST_SHORT_MSG do
7  |  if rank==request→leader do
8  |  |  if rank==root do PDMPIC_intra_bcast(buf,…) end
9  |  |  Inter_BT_bcast(buf,…,PDMPIC_intra_bcast);
10 |  else PDMPIC_intra_bcast(buf,…);
11 |  End
12 Else
13 |  if rank==request→leader do
14 |  |  Inter_BT_scatter(buf,…,NULL);
15 |  |  if rank==root do PDMPIC_intra_bcastseg(buf,…) end
16 |  |  if nbyte<BCAST_LONG_MSG &&ispof2(nodecount) do
17 |  |  |  Inter_RD_allgather(buf,…,PDMPIC_intra_bcastseg);
18 |  |  else
19 |  |  |  Inter_RING_allgather(buf,…,PDMPIC_intra_bcastseg);
20 |  |  end
21 |  else
22 |  |  do
23 |  |  |  mpi_errno=PDMPIC_intra_bcastseg(buf,…)
24 |  |  while mpi_errno==MPI_SUCCESS
25 |  end
26 end
27 return
```

Fig.7 Competitive and pipelined broadcast algorithm

图 7 竞争式流水化广播算法

但是,通过目前流行 MPI 库(比如 MPICH2[13]和 Open MPI[8])的基础结构高效实现竞争式流水化广播算法存在两方面的困难:第一,如何高效实现“竞争”;第二,由于各个节点上的引导进程是由竞争机制动态确定的,引导进程之间不知道其他节点引导进程的进程号,如何进行节点间通信?因此,针对目前 MPI 库的不足,我们设计实现了一种“MPI 加速器”,并在“MPI 加速器”的框架中设计实现了如图 7 所示的竞争式流水化广播算法.下文简称 MPI 加速器为 MPIActor.

图 8 示例了基于 MPIActor 的 MPI 并行任务的执行原理及其与标准 MPI 库的结合关系.在基于 MPIActor 的 MPI 并行任务中,MPI 进程是一个逻辑单位;实际运行时,MPI 进程以线程为系统执行单位运行,且与节点上属于相同并行任务的 MPI 进程都运行在同一个容器进程中.MPIActor 基于标准的 MPI 库,并用更适合于混合线程、进程间通信的算法重载其 MPI 通信接口,向用户 MPI 进程提供更高效的通信能力.MPIActor 在实现 MPI 通信接口的算法时,可以通过标准 MPI 中的点对点通信接口支持节点间通信.即通过 MPIActor 的基础结构支持一个节点中 MPI 进程发送的消息可以被另一个节点中的任意 MPI 进程所接收,这解决了前文所述实现高效竞争式流水化广播算法的难点 2.

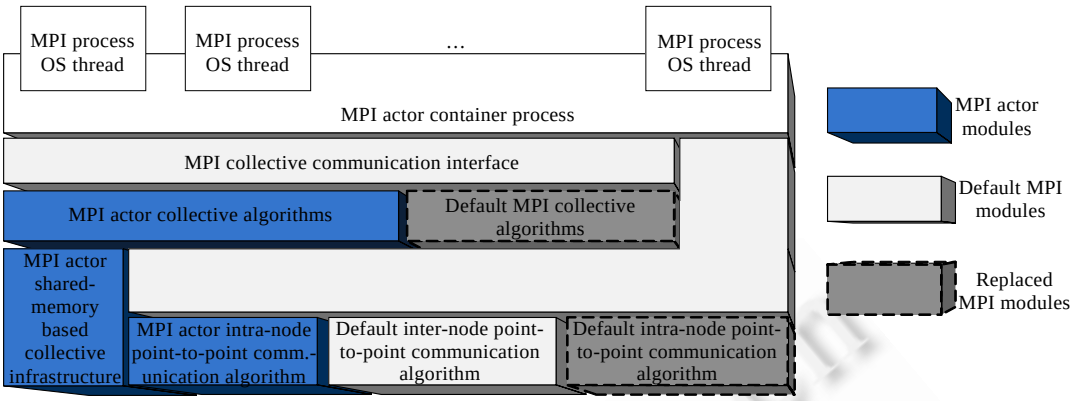

Fig.8 MPIActor provides user program with more effective intra-node point-to-point communication and collective communication based on standard MPI library

图 8 MPI 加速器基于 MPI 库向用户程序提供更高效的节点内点对点通信和集合通信

MPIActor 通过“共享内存集合通信支持部件(shared-memory based collective infrastructure)”对 MPIActor 中的集合通信算法进行高效支持,通过这个部件,各个节点内属于同一通信域的 MPI 进程都通过通信域对象(communicator)共享同一集合通信请求队列,如图 9 所示.在这种机制下,容易设计如图 10 所示的高效竞争算法.在算法中,comm_ptr→crq 表示集合通信请求队列,例程 find_unsolved_req 能够从集合通信请求队列中找到第 1 个未处理的集合通信请求.通过该算法,同一次通信每个节点中的相关 MPI 进程通过对一个简短临界区的访问得到集合通信请求对象并确定引导进程,以较为简洁的方式实现了“竞争”,解决了前文所述实现高效竞争式流水化广播算法的难点 1.

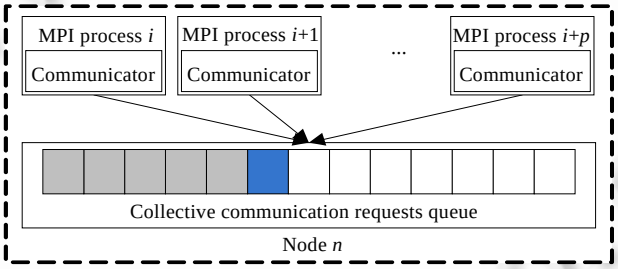

Fig.9 Same node MPI processes share one collective communication requests queue

图 9 相同节点的 MPI 进程共享同一集合通信请求队列

```
1  Algorithm. get_coll_request_ptr
   Input: type/*request type*/, comm_ptr/*communicator*/;
   Output: req/*collective communication request*/.
2  Mutex_lock(comm_ptr→crq→mutex);
3  req=find_unsolved_req(comm_ptr→crq,type);
4  if req==NULL do
5  |  req=new_req(type);
6  |  enqueue(req,comm_ptr→crq);
7  |  req→leader=isrootnode? root: myrank;
8  End
9  Mutex_unlock(comm_ptr→crq→mutex);
10 return req;
```

Fig.10 Competitive algorithm in get_coll_request_ptr

图 10 get_coll_request_ptr 中的竞争算法

4 实验与结果

为了验证前文提出的优化方法,我们在真实平台下对第 3 节所述的 CP 广播算法进行了两类性能评测实验:第 1 类实验参照文献[1,10,11]中的性能评测方法进行,属于微基准测试(micro benchmark),目的在于评价单一通信操作在某确定条件下的性能;第 2 类实验利用了两个常用的算法例程进行性能评测,对比采用 MVAPICH 广播和采用 CP 广播时的程序耗时,目的在于评价 CP 广播在复杂应用场景中的性能.

本文的实验环境是一个集群系统中的 16 个节点,每个节点内有两颗 Intel Xeon 5150 四核处理器和 8GB 内存,处理器频率为 2.66GHz,节点间用 4X DDR Infiniband (20Gbps)互联,实验平台的操作系统使用 Red Hat Enterprise Linux4,MPI 库采用 MVAPICH2 1.2.

4.1 微基准测试

评价 MPI 广播通信在 UPA 模式下性能的微基准测试方法如图 11 所示,该测试方法参照文献[1,10,11]中的实验方法来设计,最大平均通信耗时大小是用来衡量算法性能好坏的标准,其中,*delay* 函数用于产生延迟,测试算法在执行 MPI 广播前以 0~*cc*max 之间的随机数值为参数调用 *delay* 函数模拟进程的随机到达,*cc*max 对应于 *delay* 函数产生到达区间宽度延迟的输入值.本节将利用这种方法分两部分对竞争式流水化广播算法进行性能评测和对比:第 1 部分评测 CP 方法对广播算法受 UPA 影响的改善,与之比较的是通过普通流水化方法优化的相应算法,后者同样基于 MPIActor 实现;第 2 部分将对比竞争式流水化广播算法相对于 MVAPICH2 1.2 中广播算法在 UPA 模式下的性能提高.

```
…
for (i=0; i<nrepeat; i++) {
   cc=ccmax==0?0: rand()% ccmax;
   MPI_Barrier(comm);
   delay(cc);
   t0=MPI_Wtime();
   MPI_Bcast(…);
   t1=MPI_Wtime()−t0;
   tt+=t1;
}
tt=tt/nrepeat;
MPI_Reduce(&tt,&t,1,MPI_DOUBLE,
            MPI_MAX,rc t,comm);
…
```

```
double delay(int circle_ ount)
{
   double A=B=C=D=1. ;
   for (int i=0; i<circle_ ount; i++)
   {
      A=(B+C)*0.618+i;
      D=(B*C)+C;
      B=(A+D)/1.08777;
      C=A*0.3456;
      B=B−floor(B);
      C=C−floor(C);
   }
   return D;
}
```

Fig.11 Performance evaluation for UPA broadcast

图 11 广播通信在进程非平衡到达情况下的性能测试

4.1.1 竞争式流水化广播算法 vs.流水化广播算法

图 12~图 17 评测了由 CP 方法优化的 BT,RDAAS 和 RingAAS 广播算法在 BPA 和 UPA 模式下的性能,并与通过普通流水化方法优化的相应算法进行比较.图例中:前缀 CP 和 P 分别表示“竞争式流水化”和“流水化”;*k* 表示进程到达区间的宽度,单位为毫秒,与纵轴“平均通信耗时”单位一致;横轴表示广播消息长度,单位为字节(Byte)或千字节(KByte),其中,1K=1000.

图 12~图 15 中的结果由运行在 16 个节点上的 64 个或 128 个 MPI 进程得出,相应的每个节点上运行 4 个或 8 个 MPI 进程.图 16、图 17 中的结果由运行在 10 个节点上的 40 个或 80 个 MPI 进程得出,相应的每个节点上有 4 个或 8 个 MPI 进程.结果显示,普通的流水化广播算法受 UPA 的影响严重,通过 CP 方法可以显著地改善广播算法受 UPA 的影响.

图 12、图 14、图 16 中的结果对应节点内 MPI 进程规模为 4,图 13、图 15、图 17 中的结果对应节点内 MPI 进程规模为 8.比较后可知,节点内进程规模越大,CP 方法的优化效果就越显著.

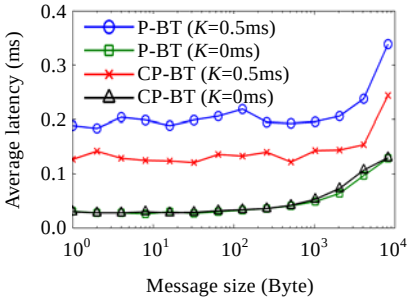

Fig.12 CP-BT broadcast vs. P-BT broadcast (MPI process number=4×16)

图 12 竞争式流水化 BT 广播 vs.流水化 BT 广播(MPI 进程数=4×16)

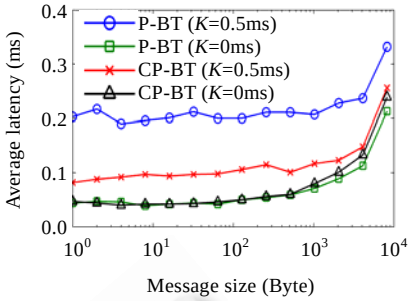

Fig.13 CP-BT broadcast vs. P-BT broadcast (MPI process number=8×16)

图 13 竞争式流水化 BT 广播 vs.流水化 BT 广播(MPI 进程数=8×16)

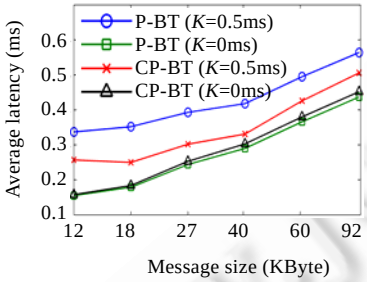

Fig.14 CP-RDAAS broadcast vs. P-RDAAS broadcast (MPI process number=4×16)

图 14 竞争式流水化 RDAAS 广播 vs.流水化 RDAAS 广播(MPI 进程数=4×16)

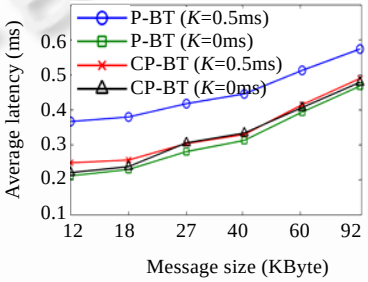

Fig.15 CP-RDAAS broadcast vs. P-RDAAS broadcast (MPI process number=8×16)

图 15 竞争式流水化 RDAAS 广播 vs.流水化 RDAAS 广播(MPI 进程数=8×16)

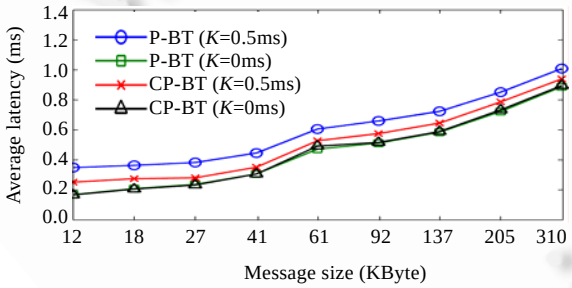

Fig.16 CP-RingAAS broadcast vs. P-RingAAS broadcast (MPI process number=4×10)

图 16 竞争式流水化 RingAAS 广播 vs.流水化 RingAAS 广播(MPI 进程数=4×10)

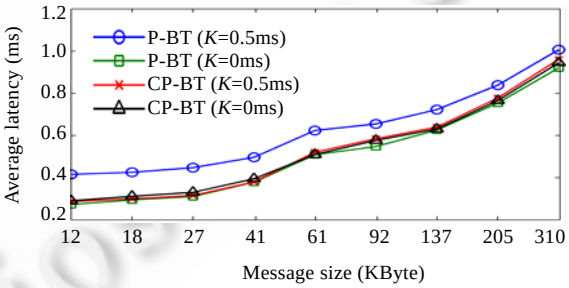

Fig.17 CP-RingAAS broadcast vs. P-RingAAS broadcast (MPI process number= 8×10)

图 17 竞争式流水化 RingAAS 广播 vs.流水化 RingAAS 广播(MPI 进程数=8×10)

4.1.2 MPIActor+MVAPICH2 1.2 vs. MVAPICH2 1.2

MVAPICH2 1.2 在广播长度为 12 288 字节到 2^{20} 字节的消息时采用了基于共享内存的流水化广播算法,对处理更小或更大的消息采用基于点对点通信的算法[14].在图 18~图 20 中,箭头指向一侧表示 MVAPICH2 1.2 在广播相应长度消息时采用基于共享内存的流水化广播算法.

图 18 与图 20 对比了 BPA 模式下的广播性能.从图中可以看出:在 BPA 模式下广播小于 2^{20} 字节长度的消息,MPIActor 中广播的性能与 MVAPICH2 1.2 中广播的性能基本相同;当广播更大的消息时,MPIActor 中广播的性能是 MVAPICH2 1.2 中广播性能的 168%~258%,这是由于 MPIActor 中高效流水化算法的贡献.

图 19 与图 21 对比了当进程到达区间为 0.5ms 时 MPIActor 中广播的性能与 MVAPICH2 1.2 中广播的性能.由图可以看出,在 UPA 模式下,MPIActor 中竞争式流水化广播的性能显著优于 MVAPICH2 1.2 中的广播算法.对比图 18 中的结果,当广播小于 2^{20} 字节的消息时,CP 方法是 MPIActor 性能优于 MVAPICH2 1.2 的主导因素,进程非平衡到达对 MVAPICH2 1.2 中的广播算法影响较大,而通过竞争式流水化方法可以较好地缓解 UPA 对广播性能的影响.在图 21 中,当广播较大的数据时,由于进程到达时间的区间值远远小于通信时间,广播性能受 UPA 的影响相对较小.对比图 20 中的结果,广播长消息时,MPIActor 中竞争式流水化广播算法的性能明显优于 MVAPICH2 1.2 中广播算法性能的主导因素是 MPIActor 中的高效流水化算法.

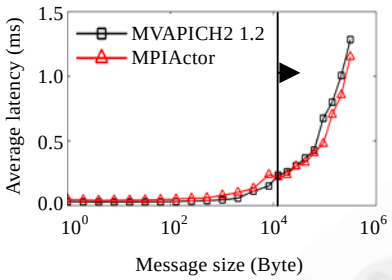

Fig.18 MVAPICH2 1.2 broadcast vs. MPIActor (MPI process number=8×16, k=0ms (short message))

图 18 MVAPICH2 1.2 的广播 vs.MPIActor 的广播 (MPI 进程数=8×16,k=0ms(短消息))

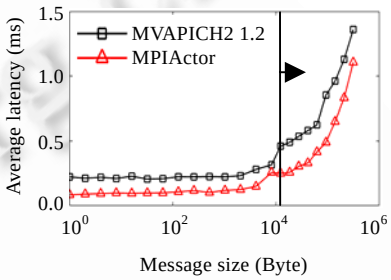

Fig.19 MVAPICH2 1.2 broadcast vs. MPIActor broadcast broadcast (MPI process number=8×16, k=0.5ms (short message))

图 19 MVAPICH2 1.2 的广播 vs.MPIActor 的广播 (MPI 进程数=8×16,k=0.5ms(短消息))

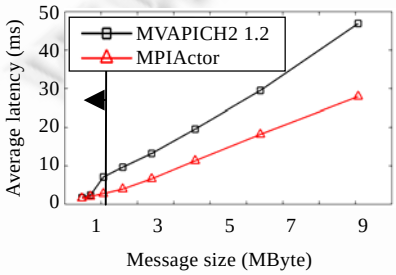

Fig.20 MVAPICH2 1.2 broadcast vs. MPIActor broadcast (MPI process number=8×16, k=0ms (long message))

图 20 MVAPICH2 1.2 的广播 vs.MPIActor 的广播 (MPI 进程数=8×16,k=0ms(长消息))

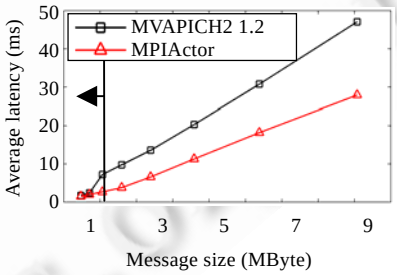

Fig.21 MVAPICH2 1.2 broadcast vs. MPIActor broadcast t(MPI process number=8×16, k=0.5ms (long message))

图 21 MVAPICH2 1.2 的广播 vs.MPIActor 的广播 (MPI 进程数=8×16,k=0.5ms(长消息))

4.2 应用测试

在本节中,我们通过两个实例 ASP 和 VCC 进一步验证 CP 方法的有效性.在实验中,我们首先对两个实例都生成了 3 个可执行程序:Xcpb,Xpb 和 Xmva,分别表示程序中的广播通过 MPIActor 中的 CP 广播、MPIActor 中的 P 广播和 MVPAICH2 中的广播完成.实验过程中,程序会记录每次广播通信的开始时间和结束时间.在主要算法执行期间,仅将这些记录保存在内存中,待程序结束时再写到文件中,写文件的时间不会被统计在程序总耗时

中.这样的数据收集方法能够基本保证统计方法不影响程序的性能,得出的数据能够真实地反映程序运行时的情况.

利用收集的数据,我们统计了应用程序运行时广播的进程到达模式,测算了它们在 16×8 个进程并行执行时的总耗时和广播通信耗时.为了使不同时间的广播具有可比性,我们对每次广播都找出了最早到达的进程 P_f,将每一个进程同上下文广播的到达时间减去 P_f 的此次到达时间,定义为相对到达时间,下文在图 22 和图 24 中都有相关引用.特别需要说明的是,在统计的进程到达模式中,我们已充分考虑了节点间时钟差异的问题.

下面将分两节分别介绍两个应用实例的实验结果.

4.2.1 实例 1:ASP(任意节点间最短路径问题)

例程 ASP 采用 Floyd-Warshall 算法并行求解“任意节点间最短路径(all-pairs shortest path)”问题[15].当 ASP 求解规模为 N 的问题时会将 $N\times N$ 的邻接矩阵按行划分,分配给各个任务,求解过程中会调用 N 次 MPI_Bcast,每次广播 N 个整数.在本文实验中,N=10240.

图 22 绘制了 128 个进程 ASPmva 中多次广播的相对进程到达时间,不同的标记表示不同上下文的广播,从 ASPcpb 和 ASPpb 中统计出来的进程到达时间与此相似.当用 128 个进程解决规模为 10 240 节点的问题时,广播的进程到达区间约为 300μs .

图 23 比较了 ASP 在采用不同广播算法时的性能,图中的数据是多次实验的平均值.从图中可以看出,以 ASPmva 为基准,采用 CP 广播的 ASP 花费在广播上的耗时比其余两个减少了 18%和 16%,计算耗时基本相同.

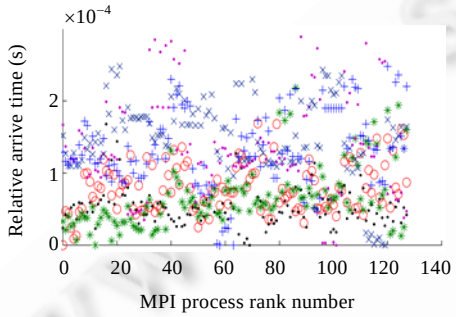

Fig.22 Stat. of process arriving time of broadcast in ASP (128 MPI processes)

图 22 ASP 中广播的进程到达时间统计(128 个进程)

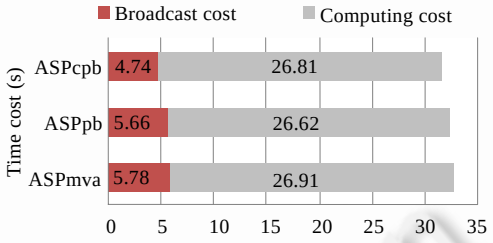

Fig.23 Performance compare of ASP by using different broadcast algorithm

图 23 ASP 在采用不同广播算法时的性能对比

4.2.2 实例 2:VCC(顶点倒塌算法解图连通分量问题)

例程 VCC 用顶点倒塌(vertex collapse)算法[16]并行求解图的连通分量问题,当求解 $N\times N$ 的邻接矩阵时,VCC 会调用 $(4+\lceil\log_2 N\rceil)\cdot\lceil\log_2 N\rceil$ 次 MPI_Bcast,每次广播 N 个整数;除此之外,还需要调用与广播同样次数的 MPI_Gather 和 $4\cdot\lceil\log_2 N\rceil$ 次 MPI_Barrier.在本文实验中,N=9216,相应地,在运行时会发生 252 次广播.

图 24 绘制了 128 个进程 VCCmva 中多次广播的相对进程到达时间,不同的标记表示不同上下文的广播;从 VCCcpb 和 VCCpb 中统计出来的进程到达时间与此相似.当用 128 个进程解决规模为 9 216 节点的问题时,广播的进程到达区间约为 200μs.

图 25 比较了 VCC 在采用不同广播算法时的性能,图中的数据是多次实验的平均值.由图可以看出:以 VCCmva 为基准,采用 CP 广播的 VCC 运行时花费在广播上的时间比其余两个减少了 24%和 17%.另外,基于 MPIActor 的 VCC 较基于 MVAPICH2 的 VCC 总体性能提高了 30%左右.这是因为,当并行度为 128 时,通信耗时占 VCC 总耗时的比重较大,为此 MPIActor 中较为高效的 MPI_Gather 做出了贡献.

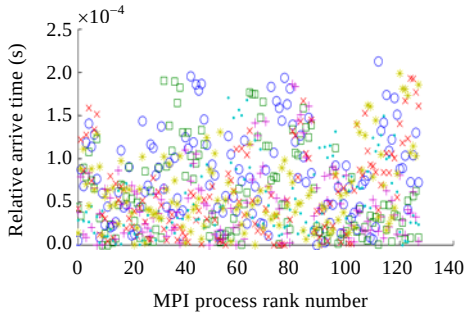

Fig.24 Stat. of process arriving time of broadcast in VCC (128 MPI processes)

图 24 VCC 中广播的进程到达时间统计(128 个进程)

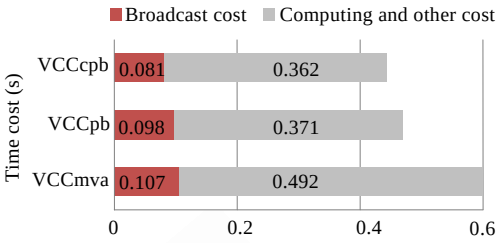

Fig.25 Performance compare of VCC by using different broadcast algorithm

图 25 VCC 在使用不同广播算法时的性能对比

5 结论与未来的工作

UPA 对传统广播算法的性能有非常大的影响,在目前流行的多核(多处理器)集群中,竞争式流水化方法可以优化传统的 MPI 广播算法,有效地减少传统 MPI 广播算法受 UPA 的影响,从而提高 MPI 广播在 UPA 模式下的性能.对基于竞争式流水线方法的广播,节点内的 MPI 进程数越多,性能受 UPA 的影响越小,在 UPA 模式下的性能也就越高.在真实环境下的微基准测试实验中,CP 广播显著地提高了 UPA 模式下的广播性能;在应用程序实验中,CP 广播的性能较 P 广播提高 16%左右,较 MVAPICH2 1.2 中的广播提高 18%~24%.所有结果都表明,CP 方法是一种针对 UPA 广播有效的优化方法.理论上,CP 方法还可以应用到优化其他 collective 通信操作.我们下一步的工作准备将竞争式流水化方法扩展到优化其他集合通信上,并结合更多的实际应用开展更广泛和深入的研究.

致谢 感谢国防科学技术大学高性能计算中心朱敏老师和赵娟老师,在进行本文实验的过程中,他们给予了很大的帮助.感谢国防科学技术大学计算机学院计算机系刘万伟博士,在进行本文理论分析的过程中,他提出了很好的建议.感谢国防科学技术大学计算机学院软件所彭绍亮博士和并行与分布处理重点实验室陈振邦博士,在本文的写作上,他们都给予了耐心的指导.

References:

[1] Faraj A, Patarasuk P, Yuan X. A study of process arrival patterns for MPI collective operations. Int'l Journal of Parallel Programming, 2008,36(6):543−570. [doi: 10.1007/s10766-008-0070-9]

[2] The MPI Forum. The MPI-2: Extensions to the message passing interface. 1997. http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html

[3] Kesavan R, Bondalapati K, Panda DK. Multicast on irregular switch-based networks with wormhole routing. In: Proc. of the IEEE HPCA. San Antonio, 1997. 48−57. [doi: 10.1109/HPCA.1997.569602]

[4] Thakur R, Rabenseifner R, Gropp W. Optimization of collective communication operations in MPICH. Int'l Journal of High Performance Computing Applications, 2005,19(1):49−66. [doi: 10.1177/1094342005051521]

[5] Patarasuk P, Faraj A, Yuan X. Pipelined broadcast on ethernet switched clusters. In: Proc. of the 20th IEEE IPDPS. Rhodes Island: IEEE, 2006. [doi: 10.1109/IPDPS.2006.1639364]

[6] Mamidala AR, Kumar R, De D, Panda DK. MPI collectives on modern multicore clusters: Performance optimizations and communication characteristics. In: Proc. of the CCGRID. Lyon, 2008. 130−137. [doi: 10.1109/CCGRID.2008.87]

[7] Watts J, Van De Gejin R. A pipelined broadcast for multidimensional meshes. Parallel Processing Letters, 1995,5(2):281−292. [doi: 10.1142/S0129626495000266]

[8] Open MPI. Open source high performance computing. http://www.open-mpi.org/

[9] Ritzdorf H, Traff JL. Collective operations in NEC's high-performance MPI libraries. In: Proc. of the 20th Int'l Parallel and Distributed Processing Symp. (IPDPS). Rhodes Island: IEEE, 2006. [doi: 10.1109/IPDPS.2006.1639334]

[10] Patarasuk P, Yuan X. Efficient MPI bcast across different process arrival patterns. In: Proc. of the 22nd Int'l Parallel and Distributed Processing Symp. (IPDPS). Miami: IEEE, 2008. [doi: 10.1109/IPDPS.2008.4536308]

[11] Qian Y, Afsahi A. Process arrival pattern and shared memory aware alltoall on InfiniBand. In: Proc. of the EuroPVM/MPI 2009. LNCS 5759, Espoo, 2009. 250−260. [doi: 10.1007/978-3-642-03770-2_31]

[12] Faraj A, Kumar S, Smith B, Mamidala A, Gunnels J, Heidelberger P. MPI collective communications on the blue gene/P supercomputer. In: Proc. of the ICS. Yorktown Heights, 2009. 489−490. [doi: 10.1145/1542275.1542344]

[13] MPICH2. http://www.mcs.anl.gov/mpi/mpich2

[14] MVAPICH. http://mvapich.cse.ohio-state.edu

[15] Kielmann T, Hofman RFH, Bal HE, Plaat A, Bhoedjang RAF. MagPIe: MPI's collective communication operations for clustered wide area systems. In: Proc. of the PpoPP. Atlanta, 1999. 131−140.

[16] Chen GL. Parallel Algorithm Practice. Beijing: Higher Education Press, 2004. 396−398 (in Chinese).

附中文参考文献:

[16] 陈国良.并行算法实践.北京:高等教育出版社,2004.396−398.

刘志强(1981－),男,山东临朐人,博士生,助理研究员,CCF 学生会员,主要研究领域为并行计算,高效 MPI 库设计.

卢风顺(1982－),男,博士生,助理研究员,CCF 学生会员,主要研究领域为 GPU 计算,并行算法设计与性能优化.

宋君强(1962－),男,研究员,博士生导师,主要研究领域为并行算法,数值天气预报.

徐芬(1983－),女,工程师,主要研究领域为自然语言处理,并行计算.