T-NBC:透明的 MPI 非阻塞集合操作

李 强[1),2),3)] 孙凝晖[1),2)] 霍志刚[1)] 马 捷[1)]

[1)](中国科学院计算技术研究所高性能计算机研究中心 北京 100190)

[2)](中国科学院计算机系统结构重点实验室 北京 100190)

[3)](中国科学院研究生院 北京 100049)

摘 要 在不修改应用程序的前提下,在 MPI 通信库中将阻塞的集合操作转化为非阻塞的实现可以将集合通信与紧跟在集合操作之后的计算重叠起来,从而提高应用的性能.在应用中,集合操作之后的计算包括集合通信无关的计算和集合通信相关的计算两类.集合通信可以与前者很好地重叠;由于后者需要访问通信数据,与后者的重叠和集合通信中多个集合子消息的通信顺序密切相关.在该文中,我们实现了对应用透明的非阻塞集合操作T-NBC(Transparent Non-Blocking Collective operations).T-NBC 不但将集合通信与集合通信无关的计算充分重叠起来,而且为了进一步增大集合通信与集合通信相关计算的重叠,它可根据应用访问多个集合子消息的顺序赋予这些子消息不同的通信优先级.微基准测试显示,T-NBC 可以将绝大部分的集合通信与集合操作之后的计算重叠起来.在 NPB(NAS Parallel Benchmarks)测试 FT(Fourier Transform)和 IS(Integer Sort)中,尽管集合操作之后的计算主要为集合通信相关的计算,但很大部分的集合通信时间被重叠,它们的性能分别提高了 5%和 36%.

关键词 透明;非阻塞;集合操作;重叠;优先级

中图法分类号 TP302 **DOI 号**: 10.3724/SP.J.1016.2011.02052

T-NBC: Transparent Non-blocking MPI Collective Operations

LI Qiang[1),2),3)] SUN Ning-Hui[1),2)] HUO Zhi-Gang[1)] MA Jie[1)]

[1)](*Institute of Computing Technology, Chinese Academy of Sciences, Beijing* 100190)

[2)](*Key Laboratory of Computer System and Architecture, Chinese Academy of Sciences, Beijing* 100190)

[3)](*Graduate University of Chinese Academy of Sciences, Beijing* 100049)

Abstract Without modifying MPI applications, transparently translating MPI collective operations into non-blocking ones in communication libraries can overlap collective communication with the computation following the operations and benefit most current applications. In applications, the following computation includes communication-unrelated computation(CURC) and communication-related computation(CRC). CURC is easier to overlap with collective communication; however, CRC need access communication data and is more difficult to overlap with collective communication. In the paper, we propose transparent non-blocking collective operations(T-NBC). It can obtain the overlap between collective communication and following communication. Besides the overlap with CURC, it improves the overlap with CRC by transmitting collective messages with different priorities according to their accessed sequence in applications. Evaluations of micro-benchmark demonstrate that a large potential overlap between collective communication and following computation can be obtained. In FT(Fourier Transform) and IS(Integer Sort) of NPB (NAS Parallel Benchmarks), even following computation dominated by CRC, a large portion of collective communication is overlapped. Their performance is respectively improved by 5% and 36%.

Keywords transparent; non-blocking; collective communication; overlap; priorities

收稿日期:2011-08-29;最终修改稿收到日期:2011-09-19.本课题得到国家"八六三"高技术研究发展计划项目"曙光 6000 千万亿次高效能计算机系统研制"(2009AA01A129)资助.**李 强**,男,1983 年生,博士研究生,主要研究方向为高性能计算机通信等.E-mail:liqiang@ncic.ac.cn.**孙凝晖**,男,1968 年生,博士,研究员,博士生导师,主要研究领域为计算机体系结构、高性能计算和分布式操作系统等.**霍志刚**,男,1978 年生,副研究员,主要研究方向为高性能计算机容错等.**马 捷**,男,1975 年生,研究员,主要研究领域为高性能计算机通信等.

1 引　言

在高性能计算应用中，MPI 已经成为事实上的并行编程标准，它支持多种集合操作. 基于当前的 MPI 标准，所有参与集合操作的进程都必须阻塞等待集合通信的完成. 随着系统规模的不断变大，集合通信需要花费越来越多的时间，阻塞的集合操作逐渐成为阻碍应用扩展运行规模的瓶颈. 为了解决这些问题，未来的 MPI 新标准 MPI-3 引入了非阻塞的集合操作①. 通过非阻塞集合操作的支持，应用开发人员可以显式地将集合通信与计算重叠起来，从而达到“隐藏”集合通信时间的目的.

然而，当前的并行应用都是基于 MPI-1 和 MPI-2 标准中阻塞的集合操作开发的. 为了使用非阻塞的集合操作，这些应用必须重新修改它们的程序. 修改程序首先需要重新设计并行代码，其次需要在大规模系统上对修改的程序进行调试. 对大多数应用来说，这将是非常巨大的工作量. 特别是，某些应用很早以前就被开发出来，并且已经运行了很多年. 因此，在不修改应用的前提下，在通信库中实现透明的非阻塞操作 T-NBC（Transparent Non-Blocking Collective operations）是十分有意义的.

通过 T-NBC，进程在调用集合操作后立即返回，此时集合通信并没有完成或者只是部分完成. 由于当前的高性能网卡如 Infiniband 网卡支持 DMA（Direct Memory Access）②，通信过程不需要 CPU 的参与，因此剩余的集合通信可以与紧跟在集合操作之后的计算重叠执行，从而达到“隐藏”集合通信时间的目的. 为了实现 T-NBC，两方面的问题需要解决.

一方面，需要保证应用的正确性. 由于尚未完成的集合通信与集合操作之后的计算并行执行，集合通信缓冲区同时“暴露”给集合通信和集合操作之后计算，T-NBC 需要保证：在要发送的集合消息尚未完成发送之前，集合操作之后计算不会写它的发送缓冲区；或者在要接收的集合消息尚未完成接收之前，集合操作之后的计算不会读写它的接收缓冲区.

同时，由于 MPI 不支持独立的推动（Independent Progress），当集合之后的计算占用 CPU 资源时，集合通信需要获得及时的推动. 与点到点通信相比，集合通信的推动有很大不同. 点到点通信只对应一个消息. 这个消息通信的推动，可以很容易地通过异步线程或者硬件卸载（Offload）技术实现. 而集合通信对应多个消息. 比如 MPI_Alltoall 通信③，假定 N 个进程参与通信，每个进程要发送 N 个不同的消息到所有进程，同时接收来自所有进程的 N 个消息. 为了方便起见，本文将这些消息称为“集合子消息”. 多个集合子消息通信的推动很难通过简单的异步线程或者硬件卸载技术实现，它们的通信需要合理的管理和调度.

另一方面，需要提高应用的性能. 通过 T-NBC，可以获得的集合通信与计算的重叠主要取决于集合操作之后的计算. 它包括集合通信无关的计算 CURC（Communication-Unrelated Computation）和集合通信相关的计算 CRC（Communication-Related Computation）. 两者相比，CURC 不需要访问集合通信缓冲区中的数据，因此 T-NBC 可以很容易地获得集合通信与它的重叠并带来性能的提升；而 CRC 需要访问集合通信缓冲区中的数据，并且只有在集合通信缓冲区对应的集合子消息完成通信后才能访问相关数据，因此 T-NBC 可以获得的集合通信与它的重叠是十分复杂的.

当前，由于通信库与应用实现的不同，T-NBC 可以获得的集合通信与 CRC 的重叠是有限的. 一方面，通信库不清楚应用在集合操作之后对多个集合子消息的访问顺序，只能按照既定的集合通信算法进行通信. 另一方面，应用编写一般独立于具体的通信库实现，应用开发人员不应该对集合通信中多个集合子消息的通信顺序做任何假设. 因此，集合通信中集合子消息的通信顺序与应用对它们的访问顺序可能会有很大不同. 在这种情况下，虽然部分集合子消息已经完成通信，但它们可能不是 CRC 当前最期望完成的集合子消息. 因而 CRC 不能继续往下执行，T-NBC 实际获得的集合通信与 CRC 的重叠是有限的.

在本文中，我们在不修改应用的前提下实现了 T-NBC，它包括以下两个方面：

（1）关于应用的正确性，T-NBC 基于虚拟内存机制实现了对通信缓冲区访问的保护，并且基于集合通信调度表，采用异步线程推动多个集合子消息的通信.

（2）关于应用性能的提高，除了较容易获得的集合通信与 CURC 的重叠，为了进一步提高应用的性能，T-NBC 对集合通信与 CRC 的重叠进行了优化. 根据基于 trace 的方法（Trace-based）获取的应

① http://www.mpi-forum.org/
② http://www.infinibandta.org/
③ MPI 相同长度数据块的全收集散发.

用对多个集合子消息的访问顺序，T-NBC 在集合通信中赋予这些子消息不同的通信优先级，较先被访问的集合子消息被赋予较高的通信优先级. 因此，T-NBC 可以使 CRC 期待的集合子消息较早地完成通信，从而提高了集合通信与它的重叠.

通过以上措施，T-NBC 在不修改 MPI 应用程序的前提下，在 MPI 通信库中将阻塞的集合操作转化为非阻塞的实现. 它在保证应用的正确执行的前提下，将集合通信与其后面的计算重叠起来，从而很好地提高了应用的性能. 微基准测试用例(Micro-benchmark)显示，T-NBC 可以将绝大部分的集合通信与集合操作之后的计算重叠起来. 在 NPB 测试用例 FT(Fourier Transform)和 IS(Integer Sort)中，尽管集合操作之后的计算主要是集合通信相关的计算，在 ft.64.C① 中，18% 的 Alltoall 通信时间被重叠，而在 is.D.128② 中，45% 的 Alltoallv③ 通信时间被重叠；它们的性能分别被提高了 5% 和 36%.

本文第 2 节主要介绍相关工作；第 3 节分析 T-NBC 带来的集合通信与计算的重叠；第 4 节阐述 T-NBC 保证应用正确执行的工作机制；第 5 节介绍 T-NBC 提高集合通信与 CRC 重叠的方法；第 6 节给出实验结果；最后一节总结全文并展望未来工作.

2 相关工作

许多研究表明，通过通信与计算重叠的方法可以很好地提高并行应用性能[1-3]. 尽管非阻塞集合操作没有包含在当前的 MPI 标准中，文献[4-7]已经对它们进行了深入研究，并且证明使用非阻塞集合操作可以将集合通信与计算很好地重叠起来. 由于非阻塞集合操作的性能优势，它将被包含在未来的 MPI-3 标准中. 然而，当前应用都是基于 MPI-1[8] 和 MPI-2[9] 编写的，它们必须修改代码才能使用非阻塞集合操作. 在本文中，我们在不修改代码的前提下，通过 T-NBC 努力获得非阻塞集合操作所带来的重叠的好处. 此外，T-NBC 所带来的性能提高也可以作为应用采用未来 MPI-3 中的非阻塞集合操作(通过修改代码的方式)所带来的性能提高的一个下界.

在分布式共享内存的实现中，虚拟内存机制被用来识别和保护对远端内存(remote memory)的访问[10-11]. 通过虚拟内存支持，文献[12]在消息还没完成接收前就释放了阻塞的 MPI 接收调用，并且采用虚页保护的机制保证程序的正确执行. 同样借助虚拟内存的支持，文献[13]在 UPC 应用中挖掘运行时计算与通信的重叠. 这些工作主要针对点到点通信，与它们不同，我们的研究对象是 MPI 集合通信. 与点到点通信相比，集合通信需要处理多个集合子消息，更为复杂.

一个集合操作包含多个集合子消息的收发操作. 在 T-NBC 中，我们借鉴了 LibNBC[4] 中的集合通信调度表来管理和调度这些集合子消息的通信，在调度表中，包含所有待推动的集合子消息通信. 与 LibNBC 不同的是，T-NBC 不需要修改应用，并且它根据应用对集合子消息的访问顺序赋予这些子消息不同的通信优先级.

在文献[14]中，为了获得更小的延迟，小消息被赋予很高的优先级. 在本文中，我们根据应用对消息的访问顺序，赋予较先被访问的消息较高的通信优先级，从而使较先被访问的消息可以较早地完成通信.

3 T-NBC 下集合通信与计算的重叠

通过 T-NBC，应用可以将集合通信与紧跟在集合操作之后的计算重叠起来. 如引言部分所提到的，集合操作之后的计算包括 CURC 和 CRC. CURC 可以与集合通信充分重叠；而 CRC 与集合通信的重叠十分复杂. 在这部分中，我们着重分析 T-NBC 在不同的场景下带来的重叠，因此为了简化分析，我们分别对计算和集合通信进行了简化的抽象.

对于计算，假定 CURC 存在于 CRC 之前，CURC 花费的时间为 $T_{\text{CURC}}(0 \leq T_{\text{CURC}})$，显然如果没有 CURC 存在于 CRC 之前，$T_{\text{CURC}}$ 等于 0. 针对 CRC，假定 CRC 以速率 $R_{\text{CRC}}(0 \leq R_{\text{CRC}})$ 消费集合通信中的数据.

对于集合通信，假定集合通信数据的总量为 $D_{\text{comm}}(0 < D_{\text{comm}})$，集合通信需要花费的总时间为 $T_{\text{comm}}(0 < T_{\text{comm}})$，并且集合通信以速率 $R_{\text{comm}}(0 < R_{\text{comm}})$ 传输(生产)通信数据，其中通信数据的生产顺序与 CRC 对这些数据的访问顺序一致，它们满足关系 $D_{\text{comm}} = R_{\text{comm}} \times T_{\text{comm}}$.

如图 1 的 5 个子图所示，在集合通信确定的情况下，即 R_{comm} 和 T_{comm} 一定的条件下，存在五种不同的场景. 在所有子图中，集合通信通过固定斜率的线

① 64 进程测试规模为 C 的 FT 测试.
② 128 进程测试规模为 D 的 IS 测试.
③ MPI 不同长度数据块的全收集散发.

段表示.对于计算,由于 CURC 部分并不消费通信数据,它通过与时间轴并行的线段表示;而 CRC 部分需要消费通信数据,它通过不同斜率的线段表示.在场景(a)、(d)和(e)中,由于 CURC 消费的集合通信数据都为 0,为了更好区分它们,在图(a)、(d)和(e)中用置于时间轴下方的线段表示它们(这些线段只是用来表明它们并不消费通信数据).在各种的场景下,集合通信与计算的重叠如下所示:

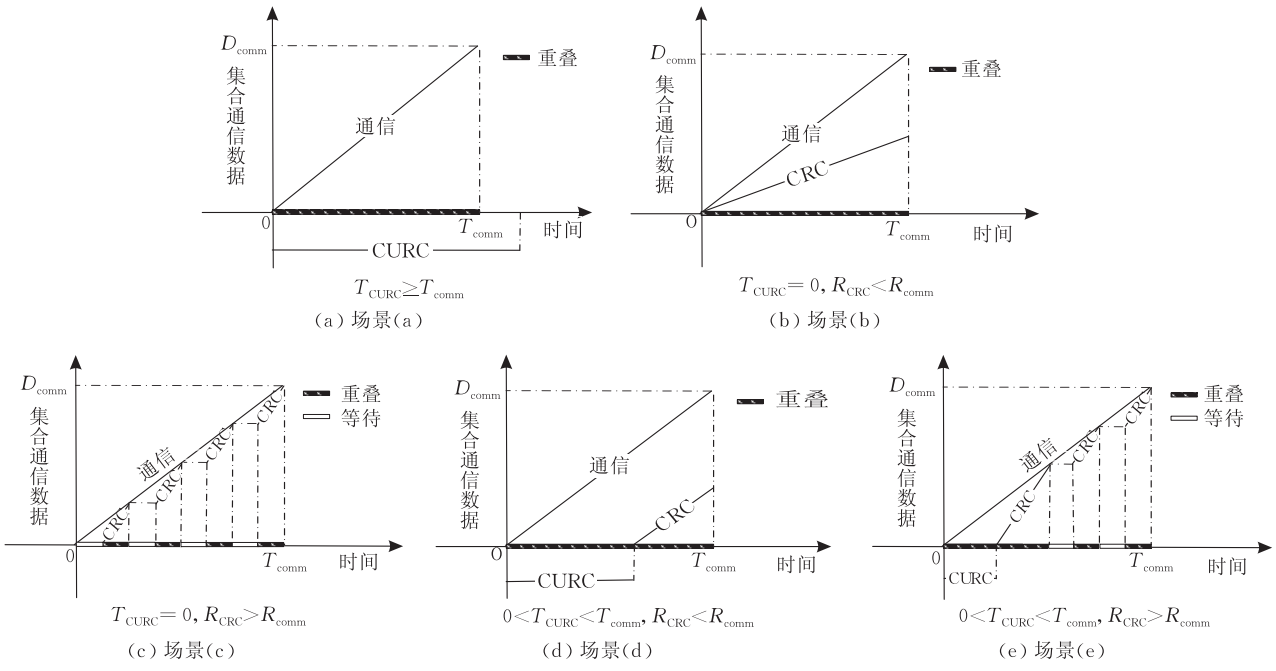


图　1

(1) 在场景(a)中,由于 CURC 足够大,并且 $T_{CURC} \geq T_{comm}$,因此可以获得重叠时间 $T_{overlap}=T_{comm}$.

(2) 在场景(b)中,由于没有 CURC 存在,并且 $R_{CRC}<R_{comm}$,因此可以获得的重叠时间 $T_{overlap}=T_{comm}$.

(3) 在场景(c)中,由于没有 CURC 存在,并且 $R_{CRC}>R_{comm}$,CRC 需要等待通信数据的到达,只有在要访问的数据到达后,它才可以继续执行,这个过程可能重复多次,因此可以获得的重叠时间 $T_{overlap}=T_{comm}\times R_{comm}/R_{CRC}$.

(4) 在场景(d)中,由于 $0<T_{CURC}<T_{comm}$,并且 $R_{CRC}<R_{comm}$,因此可以获得的重叠时间 $T_{overlap}=T_{comm}$.

(5) 在场景(e)中,由于 $0<T_{CURC}<T_{comm}$,并且 $R_{CRC}>R_{comm}$,结合(c)中所提到的分析,因此可以获得的重叠时间为 $T_{overlap}=T_{CURC}+T_{comm}\times R_{comm}/R_{CRC}$.

综合不同场景下所获的重叠,当集合通信的数据按照应用对通信数据的访问顺序进行通信时,T-NBC下集合通信与计算的重叠可以用式(1)表示.根据式(1),T-NBC 最大可以将整个集合通信重叠起来,最小与应用对通信数据的使用密切相关.同时,虽然式(1)只是基于对应用的简化抽象,但它暗示在不同的应用场景中,T-NBC 都可以带来潜在的重叠和性能提升.

$$T_{overlap}=\mathrm{Min}(T_{comm}, T_{CURC}+T_{comm}\times R_{comm}/R_{CRC}) \quad (1)$$

4　T-NBC 保证应用正确执行的工作机制

图 2 显示了应用在采用 T-NBC 后的执行流程.与原来阻塞的集合操作相比,通过 T-NBC,集合操作调用在集合通信尚未完成之前就返回,从而剩余的集合通信与集合操作之后的计算并行执行.在这

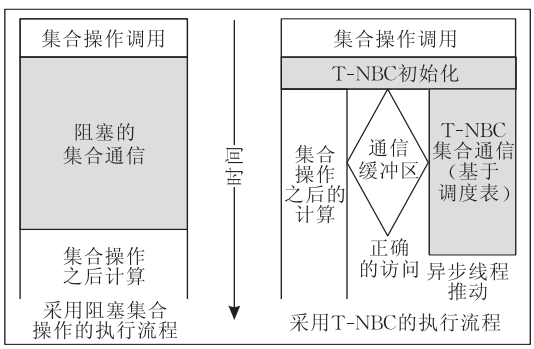


图 2　T-NBC 与阻塞集合操作执行流程比较

种情况下,集合通信缓冲区同时"暴露"给集合通信和集合操作之后的计算,因此为了保证应用的正确性,在某一通信缓冲区对应的集合子消息尚未完成通信前,需要保护这个通信缓冲区不被集合操作之后的计算错误地访问.同时,如图2所示,集合操作之后的计算会占用CPU,由于MPI没有独立的推动,集合通信需要借助异步线程获得及时的推动.

4.1 基于虚拟内存机制的通信缓冲区保护

4.1.1 基本工作机制

由于集合通信缓冲区同时"暴露"给集合通信和集合操作之后的计算,T-NBC需要保证:在与发送缓冲区对应的集合子消息尚未完成发送之前,集合操作之后的计算不会写这个缓冲区;或者在与接收缓冲区对应的集合子消息尚未完成接收之前,集合操作之后的计算不会读写这个缓冲区.在本文中,我们采用虚拟内存机制保护对通信缓冲区的访问.

操作系统支持进程通过mprotect系统调用来控制对虚拟内存页面的访问权限.针对某个虚拟内存页面,它的访问权限不是NONE,就是READ、WRITE、EXEC这三者中的某些组合.当进程执行指令访问某个虚拟页面内的内存地址时,如果指令与虚拟页面的访问权限不匹配,CPU就会产生一个保护异常,最终操作系统向进程发送一个SIGSEGV信号.

在初始化阶段,T-NBC通过mprotect系统调用分别赋予发送和接收缓冲区不同的权限.对于发送缓冲区,它被赋予只读(READ)权限;对于接收缓冲区,它被赋予不可读不可写(NONE)权限.由于一个集合操作包含多个集合子消息,为了快速查找某个虚拟地址所对应的子消息,红黑树被用来记录每个子消息地址范围(用起始地址和长度表示)与子消息的对应关系.

进程在完成T-NBC初始化后,从通信库返回,并执行集合操作之后的计算.当进程接收到SIGSEGV信号后,它将调用注册的SIGSEGV信号处理函数.信号处理函数根据异常的地址查找红黑树,并找到相应的集合子消息.如果这个子消息对应的通信已经完成,则重新赋予这个子消息所对应的缓冲区读写权限;否则,信号处理函数需要一直等待,直到这个子消息通信的完成.

4.1.2 页面保护粒度存在的问题

由于mprotect基于页表实现,缓冲区的最小保护粒度是页面.当通信缓冲区并不按照页面严格对齐时,在不对齐的页面上,不在缓冲区地址范围的内存空间也被赋予了只读READ或者NONE权限.在这种情况下,如果进程后续的执行需要访问这部分的空间,特别是当这部分空间在进程数据栈上时,进程将不能正常执行.为了避免这样的问题,在T-NBC初始化阶段,需要首先完成不对齐的页面所对应的通信,从而在相关通信完成后,可以不对这些页面进行保护.在连续的缓冲区中,不对齐的页面只可能存在于起始页和结束页,因此最多有两个页面所对应的通信需要在初始化阶段完成.随着集合通信的规模扩大,集合通信的数据增多,两个页面所对应的通信只占整个集合通信的很小一部分.

4.2 基于调度表的异步线程推动

T-NBC中集合通信的推动十分复杂.它不但需要处理多个集合子消息的收发,还需要处理它们之间的先后依赖关系,例如在MPI_Bcast基于树算法的广播中,中间节点必须在接收到其父节点的集合子消息后才能向其子节点发送集合子消息,因此需要合理地管理和组织这些子消息的通信.与此同时,MPI没有独立的推动,需要借助异步线程对这些子消息的通信进行及时地推动.

与LibNBC[4]类似,T-NBC也采用了集合通信调度表来管理多个集合子消息的通信.每一个集合子消息对应调度表中的一个消息项,消息项包含与集合子消息通信相关的所有信息:{通信缓冲区,消息长度,源,目的地,发送或接收,通信优先级}.与LibNBC不同,T-NBC中的消息项增加了调度优先级,它的作用将在第5节阐明.为了保证集合子消息间的先后依赖关系,相应的{等待}项被插入到调度表中.{等待}项用来确保它前面消息项中的集合子消息通信的完成,只有这些集合子消息的通信完成后,它后面的消息项中的集合子消息通信才能够获得执行.通过这种方式,调度表可以保证集合子消息间正确的先后依赖关系.

基于调度表,T-NBC采用异步线程推动集合子消息通信的执行.异步线程被周期性地唤醒并按照调度表执行相应的通信操作,其中通信操作的执行采用非阻塞的MPI点到点通信函数MPI_Isend或MPI_Irecv,通过这种方式,异步线程在调用这些函数后可以迅速地释放CPU,从而达到通信与计算重叠的目的.

5 T-NBC提高集合通信与CRC重叠的方法

由于T-NBC可以很容易地获得集合通信与

CURC 的重叠，为了进一步提高性能，它需要对集合通信与 CRC 的重叠进行优化. 由于 CRC 需要访问集合通信缓冲区中的数据，如 4.1 节提到的，为了保护对通信缓冲区的访问，在通信缓冲区所对应集合子消息没有完成通信之前，SIGSEGV 信号处理函数会一直等待. 因此，为了使信号处理函数尽快返回，较先被 CRC 访问的缓冲区所对应的集合子消息应该较早地完成通信. 然而，由于通信库和应用实现的差异，集合通信中多个集合子消息的通信完成顺序与应用对这些子消息的访问顺序(即 CRC 的访问集合子消息的顺序)并不一致.

为了解决这个问题，我们首先基于 trace(Trace-based)的方法获得应用对多个集合子消息的访问顺序，然后在 T-NBC 中根据获取的访问顺序赋予这些子消息不同的通信优先级，较先被应用访问的消息被赋予较高的通信优先级. 因此，较先被应用访问的消息可以较早地完成通信，进程可以较早地从信号处理函数返回并继续执行 CRC，这增加了集合通信与 CRC 的重叠.

5.1　应用对集合子消息访问顺序的获取

5.1.1　基于 trace 的方法

我们采用基于 trace 的方法获取应用对集合子消息的访问顺序，并将获取的顺序作为 T-NBC 进一步优化的依据. 在 trace 的抓取中，应用采用修改过的阻塞集合操作执行. 在修改过的实现中，在对应的集合通信完成后，集合操作函数并不立即返回. 在返回前，它首先通过 mprotect 系统调用修改发送缓冲区的权限，赋予它只读(READ)权限，修改接收缓冲区的权限，赋予它(NONE)权限，然后将地址范围与各个集合子消息的对应关系记录在红黑树中，最后记录调用结束的时间，为了减少获取时间的代价，相应的时间通过读取处理器 TSC(Time Stamp Counter)寄存器获取.

在从集合操作调用返回后，进程继续执行. 一旦进程接收到 SIGSEGV 信号后，相应的信号处理函数即被调用. 在信号处理函数中，根据异常地址可以在红黑树中查找到对应的集合子消息并记录它的访问次序，同时可以获得这个子消息的访问时间. 在这些操作完成之后，信号处理函数重新赋予子消息所对应的缓冲区读写权限并返回. 值得注意的是，trace 只记录进程对每个集合子消息的第一次访问. 在进程结束后，基于 trace 记录，可以得到所有集合子消息的访问顺序. 同时，通过集合调用结束的时间以及每个集合子消息的访问时间，也可以获得集合操作之后的计算(即 CRC)访问集合子消息的时间间隔.

5.1.2　访问顺序的选择

集合操作中的发送缓冲区和接收缓冲区各自有不同的访问顺序，需要选择唯一的一组访问顺序作为 T-NBC 中赋予集合子消息不同通信优先级的依据.

为了解决这个问题，T-NBC 在初始化阶段对发送缓冲区的保护采用其他的方法. T-NBC 在初始化阶段分配临时的缓冲区，并将原发送缓冲区的数据拷贝到临时的缓冲区中. 在拷贝完成后，T-NBC 初始化阶段结束，临时的缓冲区作为发送缓冲区参与剩余的集合通信，因此原来的缓冲区可以被集合操作之后的计算正常访问. 由于拷贝过程与部分集合子消息的通信重叠执行(见 5.2.3 节)，拷贝过程并没有增加整个集合通信的时间，并且内存的性能很高，初始化阶段的拷贝时间只占整个集合通信时间的很小一部分，在初始化阶段结束之后，仍然有很大比例的集合通信时间可以重叠. 通过这种方法，T-NBC 不再使用虚拟内存机制对原发送缓冲区进行保护，从而也就没有必要获取发送缓冲区对应的访问顺序. 因此，T-NBC 中赋予集合子消息不同通信优先级的依据是接收缓冲区中集合接收子消息的访问顺序.

5.1.3　多个进程对集合子消息的访问顺序

由于多个进程参与集合操作，每个进程都拥有一组对各自接收缓冲区中集合接收子消息的访问顺序. 假定 N 个进程参与通信，在 MPI 集合操作中，接收缓冲区中集合接收子消息的数目分为两类. 一类如 MPI_Bcast，每个进程只需要接收来自一个进程(根进程)消息，相应地，它的接收缓冲区只包含一个集合接收子消息，关于这类集合接收子消息访问顺序的处理将在 5.2.1 节阐述. 另一类如 MPI_Alltoall，每个进程需要接收来自 N 个进程的消息. 对于任意进程 P_i($\forall i(0\leq i<N)$，它的接收缓冲区包含 N 个集合接收子消息，缓冲区中的第 j($\forall j(0\leq j<N)$个集合接收子消息来自于进程 P_j. 基于 trace 的方法可以获取进程 P_i 对 N 个集合接收子消息的访问顺序如式(2)所示. 该公式表示进程 P_i 依次访问缓冲区中第 i_0 个、第 i_1 个，依次类推，直到第 $i_{(N-1)}$ 个集合接收子消息.

$$i_0,\ i_1,\ i_2,\ \cdots,\ i_{(N-1)} \tag{2}$$

其中，$i_k\in\{m\mid 0\leq m<N\}$，并且如果 $k_1\neq k_2$，则 $i_{k_1}\neq i_{k_2}$.

由于 MPI 应用多采用单程序多数据流(SPMD)的模式,在许多应用中,每个进程会使用相同的顺序访问 N 个集合接收子消息,即对于任意的进程 P_i 和 P_j($\forall j(0\leq j<N)$,在它们的访问顺序满足 $i_k=j_k$($\forall k(0\leq k<N)$.在本文中,基于访问顺序的优化主要针对所有进程采取相同的顺序访问集合接收子消息的应用.

5.2 赋予集合子消息不同的通信优先级

基于应用对集合子消息的访问顺序,在 T-NBC 中不同的集合子消息被赋予不同的通信优先级,较先被应用访问的集合子消息被赋予较高的通信优先级.有差别的通信优先级不仅存在于同一进程的多个集合子消息间,而且存在于同一结点上不同进程的集合子消息间.

5.2.1 同一进程的集合子消息的通信优先级

在同一进程的多个集合子消息间,根据集合子消息的访问顺序,赋予较先被应用访问的集合子消息较高的通信优先级相对比较简单.

这里以 MPI_Bcast 为例.在 MPI_Bcast 操作中,根进程负责广播消息,非根进程接收来自根进程的广播消息.为了获得更好的集合通信与计算的重叠,根进程和非根进程将广播消息需要切分为几个较小的集合子消息.如图 3 所示,广播消息被切分为了 4 个集合子消息.然后,通过基于 trace 的方法,可以获得非根进程对集合接收子消息的访问顺序.在图 3 中,4 个集接收子消息的访问顺序为{3,1,2,0}.针对所有非根进程采用相同的顺序访问集合接收子消息的情况,较早被访问的集合接收子消息应该赋予较高的通信优先级,同时,在根进程上,与这些集合接收消息所对应的集合发送子消息也应赋予较高

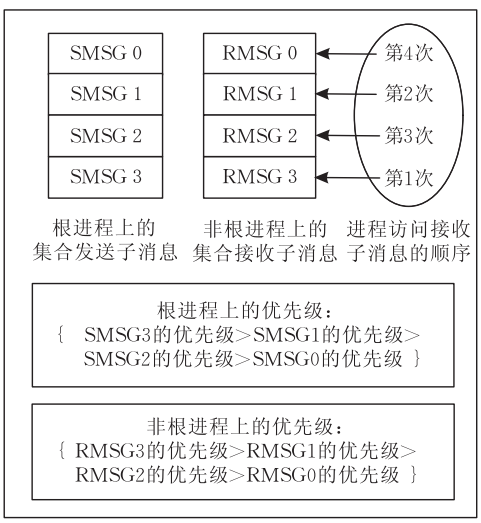


图 3 Bcast 中集合子消息的优先级

的通信优先级.图 3 分别显示了根进程和非根进程上集合子消息的通信优先级.基于不同的通信优先级,较先被访问的集合子消息可以较早地完成广播,非根进程接收到这些子消息后,可以继续执行 MPI_Bcast之后的 CRC;与此同时,较低优先级的集合子消息的广播并行执行.在根进程上,由于 T-NBC在初始化阶段将发送缓冲区的数据被拷贝到临时的缓冲区中,MPI_Bcast 之后的 CRC 可以访问原来的发送缓冲区,它也与集合子消息的广播并行执行.

当集合消息比较小时,基于切分获得集合子消息的方法并不适用.然而,随着 GPU 等加速部件的使用,进程的计算能力增强,进程间需要通信的数据越来越大,因此参与集合通信的消息也会越来越大.针对集合消息较大的情况,倘如切分获得的集合子消息的数目为 $m(m>1)$,CRC 最大可以重叠$(m-1)$个集合子消息的通信时间,因此最大的重叠比例为$(m-1)/m$.在图 3 中,当 m 为 4 时,最大的重叠比例为 75%.显然,当$(m>5)$后,随着 m 的增大,最大重叠比例的增加并不明显,同时,过多的集合子消息导致消息长度变小,消息传输的代价增大.因此,基于切分获得集合子消息的方法适宜采用较小的切分数目.

5.2.2 同一结点上不同进程的集合子消息的通信优先级

在同一结点上不同进程的集合子消息间,根据集合子消息的访问顺序,赋予较先被应用访问的集合子消息较高的通信优先级相对比较复杂.

当前的 HPC(High Performance Computing)平台大都采用多核处理器,一个结点上运行多个进程.在集合操作中,同一结点内多个进程同时使用网络,每个进程实际获得均分的网络带宽.假定每个结点包含 M 个进程,根据 LogGP[15] 模型,与只有一个进程独享网络资源相比,M 个进程均分网络带宽导致一个消息的延迟增加了将近 M 倍.为了方便下文的说明,我们假定一个进程独享网络时,一个消息的通信时间为 T;M 个进程共享网络时,通信时间为$(M\times T)$.因此,通过赋予 M 个进程的集合子消息不同的通信优先级,同一时刻同一节点内只有一个集合子消息进行通信并且独享网络带宽,从而使较先被应用访问的集合子消息可以较早的完成通信.

这里以 MPI_Alltoall 为例.针对所有进程都采用相同的顺序访问接收缓冲区中的集合接收子消息的情况,它们的访问顺序如式(2)所示,所有进程首先访问各自接收缓冲区中的第 i_0 个集合接收子消

息，这些集合接收子消息与进程 P_{i_0} 上的 N 个集合发送子消息相对应. 这意味着，每个进程只有在接收到来自进程 P_{i_0} 的集合发送子消息后，才能继续执行 Alltoall 之后的 CRC. 因此，进程 P_{i_0} 需要尽快地将它的 N 个集合发送子消息发送到所有进程. 无论 P_{i_0} 采用何种顺序发送 N 个集合发送子消息，总有一个集合发送子消息被最后发送，假定最后一个集合发送子消息发往进程 P_j.

在不采用优化的情况下，同一结点上的 M 个进程的集合子消息没有优先级差别，M 个进程同时进行消息通信并均分网络带宽. 进程 P_{i_0} 从调用集合操作到发送完成最后一个集合发送子消息到进程 P_j 需要花费的时间为 $(N\times M\times T)$. 基于 T-NBC 对缓冲区的保护，进程 P_j 上的 CRC 需要一直等待，直到这个集合发送子消息的到达进程 P_j. 然而，在这段时间内，P_j 上的集合通信并行执行，并且在 $(N\times M\times T)$ 时间后，执行完成（发送完成了 N 个集合子消息并同时接收到了来自所有进程的 N 个集合子消息）. 因此，在进程 P_j 上，Alltoall 通信与 Alltoall 之后的 CRC 实际上并没有获得重叠. 由于并行应用的性能取决于最慢的进程，在这种情况下，性能由 P_j 决定，对于整个应用来说，T-NBC 并没有带来应用性能的提升.

在采用优化的情况下，进程的集合子消息被赋予不同的优先级. 如图 4 所示，在同一个结点 0 上，进程 P_{i_0} 的集合发送子消息被赋予最高的通信优先级，它们首先被传输，并且在传输过程中独占网络. 因此它发送 N 个集合发送子消息的时间为 $(N\times T)$，从而进程 P_j 从调用集合操作到接收到来自 P_{i_0} 的集合发送子消息的时间为 $(N\times T)$. 为了使其它的结点并行执行通信操作（进程分配拓扑如图 4 所示），并且进程 $P_{i_1}, P_{i_2}, \cdots, P_{i_{(N/M-1)}}$ 的集合发送子消息在各自的结点上分别被赋予最高的通信优先级，它们与 P_{i_0} 的集合发送子消息同时被传输. 通过这种方式，在经过 $(N\times T)$ 的时间后，每个进程都获得了 (N/M) 个消息，它们分别是接收缓冲区中的第 i_0，$i_1, i_2, \cdots, i_{(N/M-1)}$ 个消息，这与应用的访问集合接收子消息的顺序一致. 因此，在接收到这些集合接收子消息后，Alltoall 之后的 CRC 可以继续执行. 与此同时，低优先级的集合子消息并行执行它们的通信，它们需要花费的时间为 $((M-1)\times N\times T)$，因此 CRC 最大可以重叠 $((M-1)/M)$ 的集合通信时间.

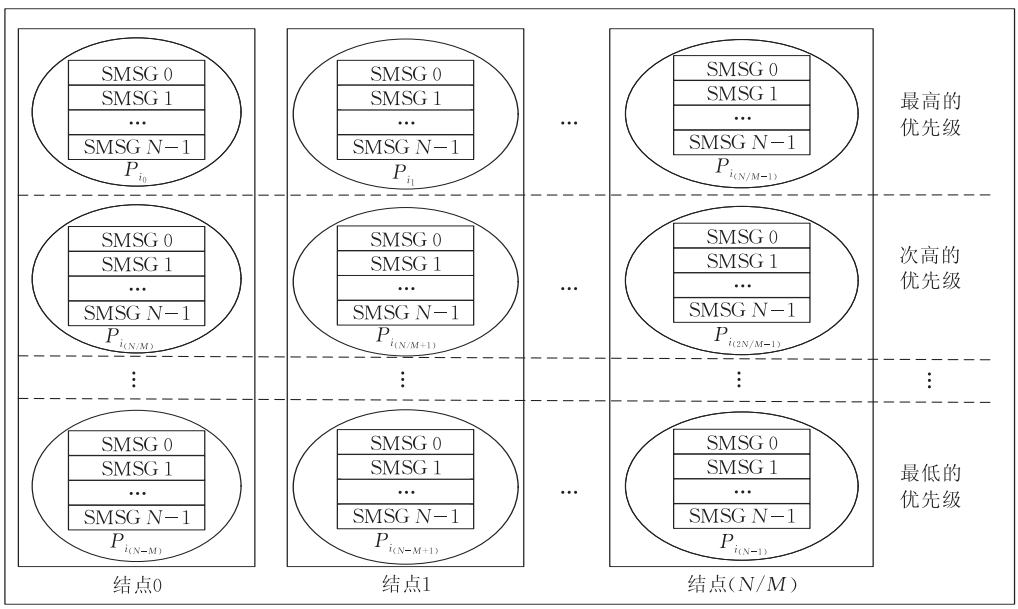


图 4　Alltoall 中集合子消息的优先级

5.2.3　初始化阶段发送缓冲区拷贝与通信的重叠

上述的两种基于集合子消息访问顺序的优化方法都需要在初始化阶段将发送缓冲区的数据拷贝到临时的发送缓冲区. 为了减少拷贝的代价，需要将拷贝过程与部分集合子消息的通信重叠起来. 在 T-NBC 初始化阶段，可以将发送缓冲区数据的拷贝过程与较高优先级的集合子消息的通信相重叠. 在拷贝过程完成后，初始化阶段结束，集合调用返回. 通过这种方法，拷贝过程并没有增加集合通信的时间，并且由于内存拷贝的性能高于网络的性能，拷贝的时间只占用集合通信时间的很小一部分. 因此，在集合调用返回后，绝大部分的集合通信仍然可以与

计算重叠.

6 性能评测

T-NBC 基于 Mvapich 库[①]实现. 实验平台是拥有 16 个结点的 Infiniband 网络机群. 每个节点使用 2 路 6 核的 2666 MHz Intel(R) Xeon(R) X5650 处理器,并且拥有一块 40 Gb/s 的 Mellanox ConnectX MT26428 HCA 网卡. 它们通过曙光 QDR HSSM 36 端口的交换机[②]连接起来. 操作系统为 centos 5.3,内核版本为 2.6.18-128.e15.

6.1 T-NBC 消息缓冲区保护开销

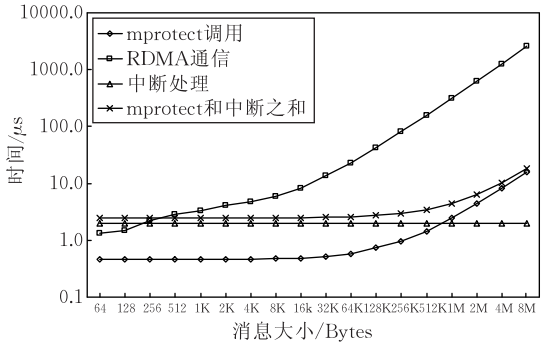
6.1.1 基于虚拟内存机制的消息缓冲区保护开销

基于虚拟内存机制的消息缓冲区保护开销主要与 mprotect 系统调用和 CPU 的段保护异常处理有关. mprotect 系统调用需要修改页表,并且会刷新 TLB 相应的表项;而段保护异常的处理需要调用中断处理程序. 图 5(a)显示这两个操作的时间以及消息通信的时间,其中由于 Infiniband 网络支持 RDMA(Remote Memory Direct Access),消息通信采用 RDMA 完成. 在现代的 CPU 处理器和操作系统中,mprotect 系统调用和中断处理都花费很小的时间. 如图 5(a)所示,在消息小于 256 KB 时,mprotect 需要修改的页表项数目小于 64(页面大小为 4KB),它所花费的时间小于 1μs. 同时,中断处理的时间为 2μs.

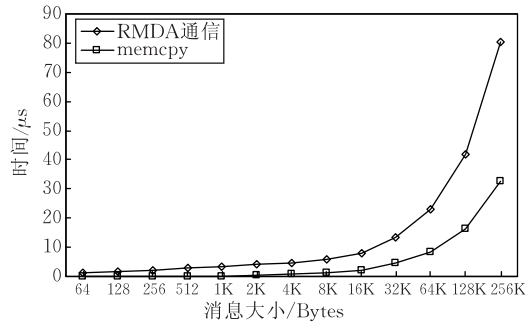
当消息长度小于 256 字节时,mprotect 系统调用和中断处理的时间之和大于消息通信的时间. 在这种情况下,由于消息缓冲区的保护开销大于消息的通信时间,因此不适宜采用 T-NBC. 然而,当消息长度大于 256B 时,消息缓冲区的保护开销小于消息的通信时间. 随着消息长度的增加,它们之间的差距越来越大,并且当消息长度大于 4 KB 时,采用 T-NBC 最大可以重叠绝大部分消息通信时间.

6.1.2 基于内存拷贝的发送缓冲区保护开销

针对优化集合通信与 CRC 重叠的情况,T-NBC 在初始化阶段需要将发送缓冲区的数据拷贝到临时的发送缓冲区. 图 5(b)分别显示了消息拷贝与 RDMA 通信所花费的时间. 从图 5(b)中可以看出,消息拷贝所花费的时间占相应消息通信时间的比例不超过 40%. 由于本实验平台中的每个结点采用双路 CPU,两个 CPU 上的进程可以并行进行拷贝;与此同时,每个结点只拥有一个网卡,两个 CPU 上的进程需要共享网络带宽,因此集合通信中所有消息的拷贝时间占它们通信总时间的比例不超过 20%. 这意味着,即使采用发送缓冲区拷贝的方法,仍然有接近 80%的集合通信时间可以重叠.



(a) 基于虚拟内存机制的缓冲区保护开销



(b) 基于内存拷贝的发送缓冲区保护开销

图 5

结合接收缓冲区基于虚拟内存机制的保护开销,在消息大于 4 KB 时,采用优化的方法的 T-NBC 仍然可以重叠绝大部分集合通信时间.

6.2 微基准测试

通过微基准测试用例,我们验证 T-NBC 在不同的场景下可以获得的集合通信与计算的重叠. 微基准测试用例的伪代码如图 6 所示.

```
MPI 集合操作;
与通信无关的计算CURC, 计算量可以调整);
For 集合操作中的每一个消息, do
    访问消息所对应的消息缓冲区并进行相关计算;
    (CRC, 计算量可以调整)
Done
```

图 6 微基准测试用例的伪代码

基准测试用例中的 MPI 集合操作采用固定的进程数目执行,并且采用固定大小的消息,这样可以保证集合操作的时间 T_{comm} 和集合通信的总数据量 D_{comm} 是固定的. 然后通过调整 CURC 和 CRC 的计算量,就可以获得如图 1 所示的不同的场景. 在本文中,微基准测试用例重点测试相对比较复杂的

① http://mvapich.cse.ohio-state.edu/
② http://www.dawning.com.cn/

MPI_Alltoall操作.它采用64个进程执行,每个结点上运行8个进程,消息的大小为1MB字节.通过测试可以得到 T_{comm} 为 268 ms,D_{comm} 为 64 MB 字节,而集合通信速率 R_{comm} 为(D_{comm}/R_{comm}),即 64MB/268 ms.表1显示了采用不同的 CURC 和 CRC 情况下,T-NBC可以获得的重叠.

表 1　不同场景下 T-NBC 获得的重叠时间

场景	原来的总时间/ms	采用 T-NBC 后的总时间/ms	重叠的时间/ms	式(3)下的重叠时间/ms
$T_{CURC}\geq T_{comm}$；$T_{CURC}=445$ ms	1060	856	204	208
$T_{CURC}=0$，$R_{CRC}<R_{comm}$；$R_{CRC}=64$ MB/350 ms	617	411	206	208
$T_{CURC}=0$，$R_{CRC}>R_{comm}$；$R_{CRC}=64$ MB/175 ms	439	332	107	115
$0<T_{CURC}<T_{comm}$，$R_{CRC}<R_{comm}$ $T_{CURC}=177$ ms，$R_{CRC}=64$ MB/350 ms	795	588	207	208
$0<T_{CURC}<T_{comm}$，$R_{CRC}>R_{comm}$； $T_{CURC}=177$ ms，$R_{CRC}=64$ MB/175 ms	618	414	204	208

在实际的 T-NBC 实现中,T-NBC 必须在完成初始化阶段对集合通信缓冲区的保护后才能返回.对于 Alltoall 操作,如 6.1 节所提到的,T-NBC 缓冲区的保护开销主要来自于发送缓冲区数据的拷贝.当消息长度为1MB时,拷贝时间约占集合通信总时间的 20%.在微基准测试用例中,通过统计 MPI 集合操作返回的时间,可以得出初始化阶段花费的时间为 60 ms,占总的集合通信时间的 22%,稍大于拷贝开销,再考虑到基于虚拟内存机制的接收缓冲区保护开销,初始化阶段开销与 6.1 节的分析基本一致.

只有 T-NBC 完成初始化返回后,所有进程才会执行紧跟在集合操作后面的计算,因此 T-NBC 可以获得最大重叠时间需要减去初始化阶段的开销 T_{init}.在本实验中,T_{init} 等于 60 ms.同时,在初始化阶段,发送缓冲区的拷贝过程与较高优先级的集合子消息通信并行执行,而最高优先级的集合子消息完成通信时间为 268 ms/8 等于 33.5 ms,小于 T_{init}.在 T-NBC 初始化阶段返回后,所有的进程都获得了最先访问的集合子消息,可以执行后面的计算.通过分析表 1 所给出的不同场景,它们都满足式(3),与式(1)相比,实际的 T-NBC 实现所获得的重叠需要减去缓冲区保护的开销.

对于 Bcast,其 T-NBC 实现与 Alltoall 类似,也需要在初始化阶段对集合通信缓冲区进行保护.因此,在采用 T-NBC 后,通过测试发现,其获得的重叠也符合式(3),为了简明起见,我们没有在本文中罗列类似结果.

$$T_{overlap}=\min(T_{comm}-T_{init},T_{CURC}+T_{comm}\times R_{comm}/R_{CRC}) \tag{3}$$

6.3　NPB 测试

在 NPB 测试用例中,FT 和 IS 的性能在很大程度上取决于它们中的集合操作性能,因此 T-NBC 可以提高它们的性能.

在 FT 中,MPI Alltoall 集合操作被使用.通过基于 trace 的方法发现,所有进程在 Alltoall 调用返回后采用相同的访问顺序访问接收缓冲区中集合接收子消息,访问顺序为 {0,1,2,3…($N-1$)},即每个进程首先访问各自接收缓冲区中的 0 号集合接收子消息,然后是 1 号集合接收子消息,依次类推,最后为($N-1$)号集合接收子消息.同时,基于 trace 的方法可以获得每个集合接收子消息的第一次被访问的时间,图 7 显示了 ft.C.32 中阻塞的 Alltoall 返回后,所有进程中 32 个集合接收子消息第一次被访问的平均时间.如图 7 所示,相对于 Alltoall 集合通信时间,集合接收子消息会在 Alltoall 返回后的很短时间内被 Alltoall 之后的计算所访问,并且访问集合接收子消息的时间间隔很短,这说明 Alltoall 之后的计算主要是与集合通信相关的计算(CRC).与 ft.C.32 类似,在 ft.C.64 和 ft.D.128 中,Alltoall 之后的计算也主要是 CRC.因此针对这些应用,T-NBC 按照集合接收子消息的访问顺序赋予集合子消息不同的通信优先级.

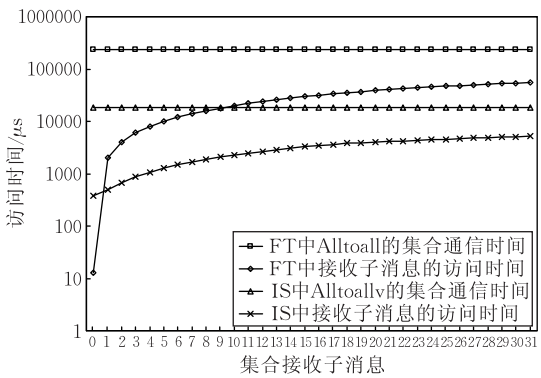


图 7　集合接收子消息的第一次被访问的时间

如图 8 所示,与原来的阻塞 Alltoall 相比,T-NBC实现了集合通信与集合操作之后的计算的

重叠,其中图 8 中所给出的时间为相应操作在 FT 中的总执行时间. 如图 8 所示,在不同的测试规模和进程数下,通过 T-NBC,Alltoall 调用的总时间不超过原来阻塞 Alltoall 调用时间的 20%,这部分时间主要用来进行 T-NBC 初始化,剩余 80%的集合通信可以与集合操作之后的计算重叠. 在采用 T-NBC 后,Alltoall 调用时间和 Alltoall 之后的计算的时间比原来的阻塞 Alltoall 调用时间和计算的执行时间减少了 10%. 这证明采用 T-NBC 后,集合通信与集合操作之后计算重叠执行,部分集合通信时间被"隐藏". 在 ft.C.32 和 ft.C.64 中,被"隐藏"的集合通信时间占集合通信总时间的比例超过 18%. 如图 10 所示,由于集合通信时间的"隐藏",不同规模和进程数下的 FT 测试的性能都获得了提高. 其中,ft.C.64 的性能被提高了 5%.

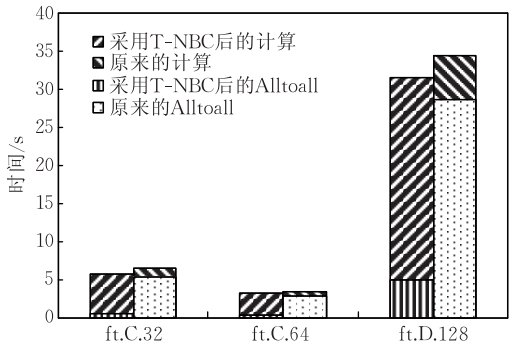


图 8 FT 采用 T-NBC 获得的重叠

在 IS 中,通过基于 trace 的方法发现,所有进程在 Alltoallv 调用返回后也采用相同的访问顺序访问接收缓冲区中的集合接收子消息,访问顺序为 $\{0,1,2,3\cdots(N-1)\}$. 同时,图 7 显示了 is.B.32 中阻塞 Alltoallv 返回后,所有进程中 32 个集合接收子消息第一次被访问的平均时间. 如图 7 所示,与 FT 类似,IS 中 Alltoallv 集合调用之后的计算也主要是 CRC.

图 9 显示了 IS 采用 T-NBC 所获得的集合通信与计算的重叠. 如图9所示,在不同的测试规模和进程数下,Alltoallv 调用的总时间不超过原来阻塞 Alltoall 调用时间的 25%. 在 is.B.32 中,超过 19%的 Alltoallv 时间被计算重叠;在 is.C.64 和 is.D.128 中,重叠比例分别为 25%和 45%. 由于 T-NBC"隐藏"了部分集合通信时间,不同 IS 测试的性能都获得了提高. 如图 10 所示,is.D.128 的性能提高了 36%.

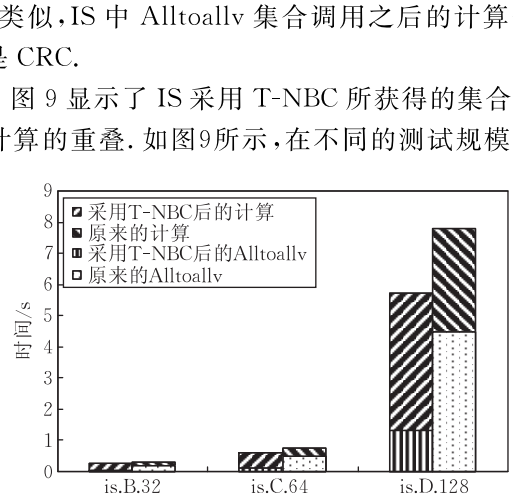


图 9 IS 采用 T-NBC 获得的重叠

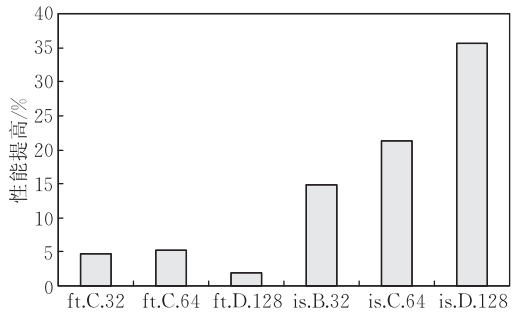


图 10 FT 和 IS 的性能提高

7 结论和未来工作

在本文中,我们在不修改应用程序的前提下实现了 T-NBC. T-NBC 通过基于虚拟内存机制的缓冲区保护方法和基于调度表的异步线程推动来保证应用的正确执行. 同时,T-NBC 可以将集合通信与集合操作之后的计算重叠起来,从而提高了应用的性能. 它不但可以获得集合通信与 CURC 的重叠,而且为了进一步提高集合通信与 CRC 的重叠,它根据应用对多个集合子消息的访问顺序赋予这些子消息不同的通信优先级. 微基准测试证明,T-NBC 可以重叠绝大部分集合通信时间. 在 NPB 测试 FT 和 IS 中,尽管集合操作之后的计算主要是 CRC,在 ft.64.C 中,18%的 Alltoall 通信时间被重叠,而在 is.D.128 中,45%的 Alltoallv 通信时间被重叠;它们的性能分别被提高了 5%和 36%.

在下一步工作中,我们将评估 T-NBC 在更多应用中的性能. Mellanox① 最近提出了基于 Infiniband 网络的集合通信硬件卸载(Offload)技术,这有利于非阻塞集合操作的实现,我们将基于这项技术对 T-NBC 进行研究.

致 谢 感谢张攀勇和张翔在各方面给予的帮助!

参 考 文 献

[1] Abdelrahman T S, Liu G. Overlap of computation and communication on shared-memory networks-of-workstations//

① http://www.mellanox.com/

Proceedings of the Cluster Computing. California, USA, 2001: 35-45

[2] Calland P-Y, Dongarra J, Robert Y. Tiling on systems with communication/computation overlap. Concurrency Practice and Experience, 1999, 11(3): 139-153

[3] Culler D, Karp R, Patterson D, Sahay A, Schauser K E, Santos E, Subramonian R, von Eicken T. LogP: Towards a realistic model of parallel computation//Proceedings of the Principles Practice of Parallel Programming. San Diego, Canada, 1993: 1-12

[4] Hoefler T, Lumsdaine A, Rehm W. Implementation and performance analysis of non-blocking collective operations for MPI//Proceedings of the 2007 International Conference on High Performance Computing, Networking, Storage and Analysis, SC07. Reno, USA, 2007: 52-61

[5] Arkady Kanevsky, Anthony Skjellum, Anna Rounbehler. MPI/RT- an emerging standard for high-performance real-time systems//Proceedings of the HICSS. Hawaii, USA, 1998: 157-166

[6] Hoefler T, Gottschling P, Lumsdaine A, Rehm W. Optimizing a conjugate gradient solver with non-blocking collective operations. Elsevier Journal of Parallel Computing (PARCO), 2007, 33(9): 624-633

[7] Hoefler T, Kambadur P, Graham R L, Shipman G, Lumsdaine A. A case for standard non-blocking collective operations//Proceedings of the PVM/MPI. Paris, France, 2007: 125-134

[8] Gropp William, Lusk Ewing, Skjellum Anthony. Using MPI: Portable Parallel Programming with the Message-Passing Interface. Cambridge, MA, USA: MIT Press Scientific and Engineering Computation Series, 1995

[9] Gropp William, Lusk Ewing, Skjellum Anthony. Using MPI-2: Advanced Features of the Message Passing Interface. Cambridge, MA, USA: MIT Press Scientific and Engineering Computation Series, 1999.

[10] Keleher P, Cox A, Swarkadas S, Zwaenepoel W. TreadMarks: Distributed shared memory on standard workstations and operating systems//Proceedings of the 1994 Winter USENIX Conference. San Francisco, USA, 1994: 115-132

[11] Keleher P, Cox A, Zwaenepoel W. Lazy Release Consistency for software distributed shared memory//Proceedings of the 19th Annual Symposium on Computer Architecture. Gold Coast, Australia, 1992: 13-21

[12] Ke Jian, Burtscher Martin, Speight Evan. Tolerating message latency through the early release of blocked receives//Proceedings of the Euro-Par Conference. Lisbon, Portugal, 2005: 19-29

[13] Iancu C, Husbands P, Hargrove P. Hunting the overlap//Proceedings of the Parallel Architecture and Compilation Techniques. Saint Louis, Missouri, 2005: 279-290

[14] Shipman G, Woodall T, Graham R, Maccabe A. Infiniband scalability in Open MPI//Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS). Rhodes, Greece, 2006: 53-60

[15] Alexandrov A, Ionescu M F, Schauser K E, Scheiman C. LogGP: Incorporating long messages into the LogP model: One step closer towards a realistic model for parallel computation//Proceedings of the 7th Annual ACM Symposium on Parallel Algorithms and Architectures. Santa Barbara, USA, 1995: 95-105

LI Qiang, born in 1983, Ph. D. candidate. His research interests focus on high performance communication.

SUN Ning-Hui, born in 1968, Ph. D., professor, Ph. D. supervisor. His main research interests include computer architecture, high performance computing and distributed OS.

HUO Zhi-Gang, born in 1978, Ph. D., associate professor. His main research interests focus on fault tolerance in HPC.

MA Jie, born in 1975, Ph. D., professor. His main research interests focus on high performance communication.

Background

This paper is supported by the National High Technology Research and Development Program (863 Program) of China under grant No. 2009AA01A129.

As HPC systems scale, hundreds of thousands of MPI processes will participate in communication. Thus, collective communication will take more and more time. According to current MPI semantics, processes must wait until collective communication is completed, so it presents applications running on a larger scale. In order to solve these problems, non-blocking MPI collective operations are presented and studied. And they will be included in future MPI standard, MPI-3. However, although non-blocking operations can overlap collective communication with computation and improve performance of applications, applications must be modified to adopt them. Most applications are developed with blocking collective operations. Rewriting applications is at a great cost to redesign and debug these parallel applications.

In this paper, we attempt to implement transparent non-blocking collective operations without modifying applications. With virtual memory support, we implement non-blocking collective operations in communication library. Calls of collective operations are quickly returned from communication library, even when the collective communication has not yet (completely) completed. In this way, collective communication is overlapped with the computation following the operations, so it greatly improves the performance of applications. Moreover, the method also determines a lower bound for the performance increase which would be expected when adopting non-blocking MPI collectives (with the cost of rewriting the applications).