基于 InfiniBand 网络的消息可扩展技术研究

彭龙根 尤洪涛 尹万旺
(国家并行计算机工程技术研究中心 北京 100080)

摘 要 InfiniBand是目前HPC系统互连的主流网络之一,其提供的可靠连接传输服务因为支持RDMA、原子操作等功能而被广泛应用于MPI等并行应用编程模型。但是支撑可靠连接所需的消息队列及缓冲区开销往往会随着并行规模的扩大而急剧增加,从而制约了应用规模的扩大。为了解决这种内存开销带来的消息可扩展性问题,先从InfiniBand传输优化方面介绍了共享接收队列和扩展可靠连接技术,然后基于并行通信模型提出了分组连接技术。通过这些技术可以将节点内存开销减少2个数量级,并且开销不会随并行规模的扩大而明显增加。

关键词 可扩展,共享接收队列,分组连接,InfiniBand
中图法分类号 TP393 **文献标识码** A

Research of Message Scalable Technology over InfiniBand Network

PENG Long-gen YOU Hong-tao YIN Wan-wang
(National Research Center of Parallel Computer Engineering & Technology, Beijing 100080, China)

Abstract InfiniBand is one of the most promising network interconnecting technologies in HPC. Its reliable connection services support RDMA, atomic operations, etc, and are widely employed in MPI and other similar parallel programming models. However, the cost of message queues and buffers for reliable connections rises dramatically when the scale of parallelism increases. As a result, it becomes the bottleneck for InfiniBand in large scale applications. To solve the problem, this paper provided shared receiving queue and extended reliable connection, then brought forward the group connection technology based parallel communication mode. On hand, the memory cost in the computing node is cut down for at least 100 times smaller. And the most amazing thing is that the cost of memory will be relatively constant when the scale of parallelism is upgraded.

Keywords Scalable, Shared receive queue, Group connection, InfiniBand

1 引言

InfiniBand是一种开放的处理器及IO互连标准,它具有高带宽、低延时等特点,目前已经成为HPC互连的主流网络之一。InfiniBand支持多种通信服务和通信语义,这些通信服务包括:可靠连接(Reliable Connection,简称RC[1])、不可靠数据报(Unreliable Datagram,简称UD[1])以及不可靠连接(Unreliable Connection,简称UC[1])等。通信语义则包括通道语义和内存语义,其中通道语义是基于传统意义上的发送/接收模型,通信双方都必须参与,且接收方需要预先投递接收请求;内存语义则是基于单边操作通信模型,它允许通信一方远程访问,另一方存储器空间而不需要远端预先投递接收请求。无论是基于哪种通信语义,通信双方均需要先在本地建立队列对(Queue Pair,简称QP[1])。典型的QP由接收队列和发送队列组成,它们分别用来组织用户投递的接收工作请求和发送工作请求。工作请求的完成情况则由InfiniBand主机通道适配器(Host Channel Adpater,简称HCA[1])写入与QP关联的完成队列上(Completion Queue,简称CQ[1])。同时,所有InfiniBand的消息缓冲区,包括QP、CQ等队列空间都需要事先锁定对应的物理内存页面才能使用,这一过程称为存储登记[1]。

目前,RC是MPI等并行语言使用最多的通信服务,因为RC支持几乎所有的InfiniBand通信语义和操作,包括send/recv、RDMA(Remote Direct Memory Access)以及原子操作等。但是RC是一种面向连接的传输服务,一个QP只能与一个远端QP建立连接并通信。也就是说,如果一个进程需要与N个远端进程通信,那么它需要创建N个QP。同时,对于send/recv操作来说,接收方为了接收远端可能的消息,每个QP都需要预先投递多个接收请求,即需要预先准备多块接收缓冲区。以N个进程问题规模为例,假设每个QP上预投递n个接收请求,每个请求接收缓冲区大小为$s1$,每个QP发送队列深度为sd,接收队列深度为rd(此处rd必须大于n),工作请求大小为$s2$。那么当所有进程之间全部建立连接时,每个进程需要创建$(N-1)$个QP,所需的空间为$(n*s1)*(N-1)+(sd+rd)*s2*(N-1)$。除此之外,还需要有CQ空间和额外的队列管理空间,考虑到CQ通常是由多

到稿日期:2012-10-19 返修日期:2012-12-30
彭龙根(1977-),男,硕士,工程师,主要研究领域为高性能互连网络,E-mail:knownature@163.com(通信作者);尤洪涛(1980-),男,硕士,工程师,主要研究领域为并行编程、并行语言;尹万旺(1980-),男,硕士,工程师,主要研究领域为并行编程、并行语言。

个 QP 所公用，且队列管理空间相比而言开销较小，故本文在评估空间需求时暂不考虑这两类空间开销。

以目前应用最广泛的 Mellanox Connectx 型号的 HCA 设备为例，其工作请求最小需要 64[2] 字节，假设上文中 $n=5$，$s1$ 大小为 8kB，发送接收队列深度均为 16，那么当问题规模为 10k 时，每个进程全连接需要的空间约为 420MB，如果是多核环境，那么单个节点所需的内存空间还需要乘以节点核数。很显然，这种基于 RC 服务的全连接通信模型会因为巨大的内存开销而严重制约应用规模的可扩展性。

针对上述因为内存开销而引起的消息可扩展性问题，本文介绍了 3 种技术。第 2 节介绍的共享接收队列(Shared Receive Queue 简称 SRQ[2])技术能够有效减少进程预投递的接收请求数目和接收缓冲区数目；第 3 节介绍的扩展可靠连接技术能够大大减少多核环境下单个节点所需的 RC 连接数目；第 4 节从并行通信模型的角度，提出了一种称为分组连接的技术。这种技术将 RC 全连接限制在一个并行组之内，并行组之间有选择地建立全连接，从而在通信模型上较好地解决了 InfiniBand 消息的可扩展性问题；第 5 节介绍了在 MPI 上应用上述 3 种方法之后，典型 NAS 测试程序的空间开销及性能对比情况。最后对 InfiniBand 消息及其可扩展性技术进行了总结。

2 共享接收队列技术

RC 模式下每个 QP 预先投递的接收请求在什么时候使用其实是未知的，即预先准备好的接收缓冲区在长度以及数目上都存在浪费。SRQ 技术引入到 RC 上的主要目的是让多个 QP 共享同一个接收队列，这意味着在创建 QP 时不再需要单独申请接收队列空间，也不再需要预先准备单独的接收缓冲区。所有的接收工作请求和接收缓冲区都集中通过 SRQ 来管理，但 SRQ 上的接收请求完成情况仍然通过与 QP 关联的 CQ 来报告。图 1 比较了使用 SRQ 前后的空间使用情况。假设 SRQ 队列深度等于 QP 个数，那么单个进程所需的空间为：$s1*(N-1)+(sd+0)*s2*(N-1)+1*s2*(N-1)$。仍然以上文的例子进行计算，当问题规模为 10k、$s1$ 大小为 8kB、发送队列深度为 16 时，单个进程所需的空间约为 90.88MB，远小于不使用 SRQ 情况下的开销。

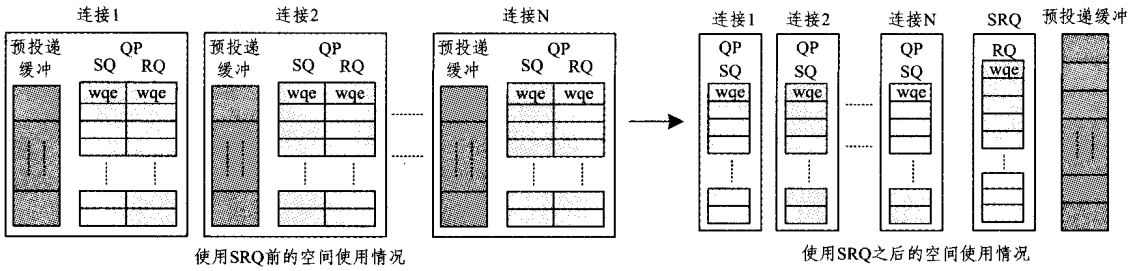

图 1 空间使用情况比较

3 扩展可靠连接技术

使用 SRQ 虽然能够大大减少进程的接收缓冲区数目，但是它仍然需要在每个进程之间都建立连接。在多核系统上，一个节点内往往运行多个进程，这就会使得单个节点上的连接数目成倍增加，内存开销也会成倍增加。此外，预投递的接收缓冲区长度实际上是预估的，并不是消息的精确长度，很多时候缓冲区大部分都是浪费的。为了解决这两个问题，扩展可靠连接(Extended Reliable Connection，简称 XRC[2])技术被引入到了 InfiniBand 传输服务中。

3.1 XRC 的特点

XRC 最开始是针对多核系统提出来的，它除了能够提供与 RC 类似的传输服务之外，其最大特点是允许发送方在发送工作请求里指定远端接收队列。如果使用了 SRQ，这意味着消息发送者可以指定消息在达到目的节点后由哪个 SRQ 来接收，因为 SRQ 是独立于连接之外能够被多个 QP 所共享的资源，所以 XRC 相当于变相地支持了一个本地 QP 可以向多个远端 QP 发送消息，也就是说，通过 XRC 一个进程只需要一条连接或者一个 QP 即可向远端节点上的所有进程发送消息。

在具体实现时，只要在初始化时将远端节点上 SRQ 与进程之间的对应关系告知发送方，那么在发送消息时选择接收 SRQ 就相当于选择了接收进程。普通 RC 模式中，一个进程需要与远端节点上的每个进程一一建立连接，连接数是随着节点核数(假设一个核上运行一个进程)增加的，使用 XRC 之后进程所需的连接数目只随节点个数增长，而不是随着核数增长。图 2 对比了普通 RC 和 XRC 模式下 2 个节点之间进程建立全连接所需要的连接数目情况。

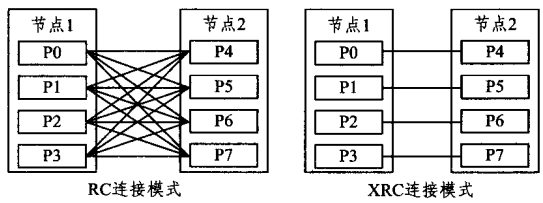

图 2 XRC 连接模式

如图 2 所示，假设每个节点有 4 个核，运行 4 个进程。RC 模式下每个节点上的连接数目为 16 个，XRC 则只需要 4 个。举个例子，在 XRC 模式中，如果进程 P0 需要发送消息给 P5，只需要在发送消息时指定接收消息的 SRQ 号为进程 P5 创建的 SRQ 即可，尽管消息传递使用的是 P0 到 P4 的通道，但是实际上消息到达节点 2 后的接收者是 P5 进程所属的 SRQ，这样 P5 进程自然就收到了 P0 进程的消息，反之亦然。

除了能减少多核系统上的连接数目之外，XRC 还有一个好处是能够减少 SRQ 上预投递接收缓冲区的浪费。在普通 RC 模式里，预投递的接收缓冲区长度是固定的，因为接收方无法事先知道发送消息的长度，所以预先准备的缓冲区长度总是留有足够冗余。比如，在 MPI 实现中 send/recv 消息最大长度设置为 8kB，超过 8kB 的用户消息通过 RDMA 操作传送，小于 8kB 的用户消息和所有 MPI 自身控制消息都用 send/recv 传送。这样所有的预投递接收缓冲区长度都是

8kB，但是实际上很多控制消息和用户消息都是远小于 8kB。但在 XRC 模式下就可以更加精细地控制接收缓冲区长度。因为发送方可以选择接收者，而发送方又事先知道消息的准确长度，所以在具体实现中，只需要在接收方准备多个长度不同的接收缓冲队列[3]，分别投递到多个 SRQ 上，比如 SRQ1 上缓冲区长度为 512B，SRQ2 上长度为 1kB，以此类推。发送方则根据消息长度选择大小合适的 SRQ 来接收消息，这样可以更有效地减少空间浪费。

3.2 XRC 内存开销

假设系统节点个数为 m，每个节点上的核数为 c，那么全连接 RC 模式下单个进程的 QP 数目为 $m*c-1$，单节点的 QP 个数为 $(m*c-1)*c$；而 XRC 模式下单进程 QP 数目为 m，单节点 QP 个数为 $m*c$。在评估 XRC 内存开销时考虑两种情况，一种是所有接收缓冲区长度相同，均为 $s1$；另一种是设置多种不同长度的接收缓冲区，为了简单起见，假设为 6 种长度，依次为 $s1$、$s1/2$、$s1/4$、$s1/8$、$s1/16$、$s1/32$，每种长度个数一样。仍然假设 QP 队列深度为 sd，工作请求大小为 $s2$，那么当所有接收缓冲区长度相同时，使用 XRC 单进程的空间需求为：$XRC\_S=m*s1+sd*s2*m+s2*m$，节点空间需求为：XRC_S*c。使用不同长度的接收缓冲区时单进程的空间需求为：

$$XRC\_M=((1+1/2+1/4+1/8+1/16+1/32)/6)*s1*m+sd*s2*m+s2*m$$

单节点空间需求为：XRC_M*c。

表 1 列出了当 $m=1000$，$c=8$ 和 16 时的空间需求值，其中 RC 是不带 SRQ 的普通 RC 模式，RC_SRQ 是带 SRQ 的 RC 模式，XRC_S 是接收缓冲长度一样的 XRC 模式，XRC_M 是设置 6 种不同长度缓冲的 XRC。

表 1 RC 与 XRC 内存开销比较

模式	单个进程内存开销		单个节点内存开销	
	m=1000，c=8	m=1000，c=16	m=1000，c=8	m=1000，c=16
RC	336MB	672MB	2688MB	10.75GB
RC_SRQ	72.7MB	145.4MB	581MB	2.326GB
XRC_S	9.088MB	9.088MB	72.7MB	145.4MB
XRC_M	3.71MB	3.71MB	29.6MB	59.2MB

从表 1 可以看出，与 RC_SRQ 模式相比，采用 XRC_M 内存开销最大，减少了将近 40 倍；与 RC 模式相比，其内存开销更是减少了 2 个数量级。

4 分组连接技术

上文讨论的空间需求均是基于进程之间静态全连接的通信模型，从理论上来说，只要是静态全连接就必然会存在可扩展性问题，即便是使用 SRQ 和 XRC 技术，内存开销仍然会随问题规模呈线性增长。

4.1 分组连接通信模型

要彻底解决可扩展性问题，必须采用更为灵活的通信模型。比如动态连接模型，在初始化时进程之间并不预先建立连接，而是在运行过程中根据实际通信需求动态建立连接，通信完毕后再关闭连接。通过动态连接可以较好地解决可扩展性问题，但是这种做法实际上是以损失性能为代价的。为了在性能和内存开销上找到平衡点，本文提出了一种静态和动态相结合的通信模型，称之为分组连接技术。

分组连接的主要思想是静动结合。所谓静，是指将所有进程划分成多个并行组，组内进程之间建立全连接关系，同时所有组内序号相同的进程之间建立全连接。所谓动，是指如果各个组之间有进程需要通信，那么先通过组之间的静态连接握手，然后在通信双方进程之间建立专用连接。图 3 是分组连接的通信模型。

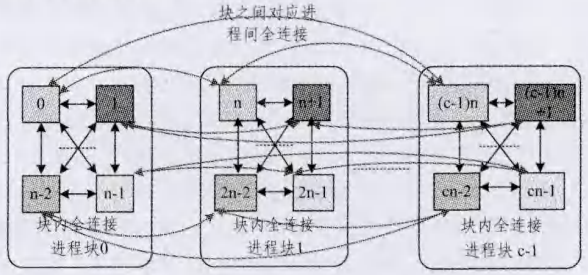

图 3 分组连接通信模型

如图 3 所示，每个进程组中进程个数为 n，一共有 c 个进程组之间的静态连接关系。其中进程组内的 n 个进程全连接，所有组内序号相同的进程之间全连接。分组连接能保证任意两个通信进程之间有直接的或者间接的连接通路，并且在具体应用中，分组连接还可以灵活使用，比如组的大小可以根据网络拓扑或者问题实际通信模型来改变。组之间的全连接还可以为组内其他进程代理消息，以避免动态连接等。总之，分组连接的使用是非常灵活的，可以根据具体应用特征制定最优的解决方案。

4.2 分组连接内存开销

与静态全连接相比，分组连接所需的连接数目不会随着并行规模的扩大而线性增长，较好地解决了消息的可扩展问题。以图 3 的结构为例，假设组大小 $n=4096$，每个节点核数为 4，整机规模为 320000 个核。那么可以将其划分为 79 个组，每个组内包含 1024 个节点，运用 XRC 技术后每个进程所需建立的连接数为 1024+78=1102 个，即每个进程需要申请 1102 个 QP，这些 QP 可以共用一个 SRQ，假设每个 SRQ 接收缓冲区长度为 8kB，每个进程需要预先投递 1102 个接收缓冲。如果 QP 发送队列深度为 16，工作请求长度为 64 字节，那么每个进程需要的空间为：

$$1102*8192+1102*16*64+1102*1*64=10.22MB$$

即便是并行规模扩大到 100 万，每个进程也只需要 1268 个连接，内存大约为 11.77MB，如果使用不同长度的 SRQ 接收缓冲，所需的内存还会更少。

5 应用与性能分析

上述 3 种 InfiniBand 消息可扩展技术目前已经应用在“神威蓝光”计算机系统 MPI 语言上。在具体实现时，以 OSU 发布的 mvapich-1.2 版本为基础进行了代码修改，即每 8192 个 QP 共用一个 SRQ，每个 SRQ 上预投递的接收请求数量等于 8192 个。接收缓冲长度默认都为 8kB(可根据环境变量调整)，超过 8k 的用户消息用 RDMA 发送，小于 8kB 的用户消息和 MPI 控制消息均使用 send/recv 模式。XRC 使用模式与上文所述一样，每个节点一个连接。分组大小默认为 4096 进程，低于或者等于 4096 进程时使用全连接。

为了能对应用性能和空间需求有更准确的评估，在“神威

(下转第 120 页)

常量参数化优化使能编译器做更深入的针对性优化，通过分级数据缓存技术提高程序访存性能，并进行了各种循环变换优化。在国家超算长沙中心"Tianhe-1A"并行计算机上进行了性能测试，结果表明，这些优化可有效提升程序的单机性能以及完整并行程序的性能，针对100万网格点二维翼型NACA0012算例，串行程序计算性能提高约22.2%～28.9%；针对1.12亿网格点三角翼算例，并行程序性能提高约13.9%～20.2%。这些优化保持了原程序的结构特点与灵活性，没有在代码中使用平台相关的编程技术或指令集，具有良好的可移植性。论文还通过微体系结构级的性能测试，分析了这些性能优化技术取得效果的内在原因。本文对于从事高性能计算应用软件开发与优化的科研人员具有一定的参考价值。

参 考 文 献

[1] Cosentino G B. Computational Fluid Dynamics Analysis Success Stories of X-plane Design to Flight Test [R]. NASA/TM-2008-214636

[2] Resch M M, Küster U. HPC Processor Technologies and Their Impact on Simulation[J]. Computational Science and High Performance Computing IV, Notes on Numerical Fluid Mechanics and Multidisciplinary Design, 2011, 115: 17-28

[3] 金君，乔楠，梁德旺. NAPA软件的并行优化[J]. 数值计算与计算机应用，2008，29(1)：65-72

[4] Gropp W D, Kaushik D K, Keyes D E, et al. Latency, Bandwidth, and Concurrent Issue Limitations in High-Performance CFD[C]//Proceedings of the First M. I. T. Conference on Computational Fluid and Solid Mechanics. Cambridge, MA, 2001

[5] 高瑞泽，于剑，阎超. 基于Cache友好方法的数值计算代码优化[J]. 计算机工程，2010，36(5)：7-9

[6] http://www.intel.com/software/products/vtune/ [OL]. 2012-05-30

(上接第106页)

蓝光"计算机系统上用NAS基准测试程序对应用性能进行了实测。"神威蓝光"计算节点配置为：每个节点4核，频率1GHz；内存4GB；InfiniBand QDR网络。表2列出了NAS测试包中消息比重较大的课题FT和LU在不同并行规模下的运行时间和消息内存开销情况。其中，测试课题规模为E规模；运行时间单位为s，内存为单个进程所需的队列和预投递接收缓冲内存开销，单位为MB。RC_SRQ为使用普通RC加SRQ的方式，XRC_S_G是使用XRC加SRQ且运用分组连接的方式，两者均使用8kB长度的接收缓冲区。

表2 性能与内存开销对比(时间单位为s，内存单位为MB)

模式/规模		2048进程		4096进程		8192进程		16384进程	
		时间	内存	时间	内存	时间	内存	时间	内存
FT	RC_SRQ	619.38	18.6	584.91	37.2	318.24	74.4	193.87	148.8
	XRC_S_G	620.21	4.65	585.06	9.3	325.52	9.315	198.68	9.344
LU	RC_SRQ	1111.1	18.6	548.95	37.2	311.62	74.4	216.4	148.8
	XRC_S_G	1111.5	4.65	549.65	9.3	316.40	9.315	220.23	9.344

从表2数据可以看出，分组连接与静态全连接相比会有一定的性能损失，这主要是因为在课题运行过程中需要临时握手、创建QP、建立连接等，这些操作都需要进入核心，带来了用户程序与核心之间的切换开销，但是我们认为，这种微小的性能损失与因空间需求的大幅减少而带来的可扩展性好处相比是值得的。

结束语 内存开销引起的可扩展性问题一直是困扰InfiniBand并行规模扩大的难题，尤其是多核系统的出现使得矛盾更加突出。本文首先介绍了基于InfiniBand消息传输服务层的两种可扩展技术：SRQ和XRC。通过这两种技术的结合，能够将单个进程所需的内存开销减少2个数量级，尤其是XRC技术在多核系统上的效果尤为明显。但是，SRQ和XRC并不能从根本上解决可扩展性问题，节点内存开销仍然会随着并行规模的增长而线性增加。

分组连接技术则是基于上层并行通信模型的可扩展性技术，它在静态全连接和动态连接之间进行了折中，使得其既具有静态全连接的性能优势，又具有动态连接的空间优势。采用分组连接技术之后，消息所带来的内存开销主要取决于组内进程个数，而不再随着并行规模线性增长，较好地解决了可扩展性问题。除此之外，分组连接还可以根据网络拓扑和应用课题的通信特征灵活配置，做到在性能和内存开销上的最佳平衡。

当然，解决可扩展性问题的技术不仅仅只有本文介绍的这些，比如MPI[4,5]有基于InfiniBand UD传输服务实现的版本，UD传输服务支持一个QP与多个QP通信而不需要建立连接，这些同样可以解决可扩展性问题。但是UD是不可靠的，且不支持RDMA操作，消息大小受限于网络MTU(Maximum Transmission Unit)[2]，所以在大规模科学计算中应用较少。本文介绍的3种技术均基于RC传输服务，其最大并行规模在"神威蓝光"计算机系统上实测已经达到了数十万核，较好地解决了InfiniBand消息可扩展性问题。

参 考 文 献

[1] InfiniBand Trade Association. InfiniBand Architecture Specification[OL]. http://www.infinibandta.com

[2] Mellanox Technologies. Connectx Family Programmer's Reference Manual[OL]. http://www.mellanox.com/docs/

[3] Koop M J, Sridhar J K, Panda D K. Scalable MPI Design over InfiniBand using eXtended Reliable Connection[C]//IEEE International Conference on Cluster Computing. 2008: 2-4

[4] Yu Wei-kuan, Gao Qi, Panda D K. Adaptive Connection Management for Scalable MPI over InfiniBand[OL]. http://cse.ohio-state.edu, 2006-02-08

[5] Friedley A, Hoefler T, Leininger M L, et al. Scalable High Performance Message Passing over InfiniBand for Open MPI[OL]. http://cs.indiana.edu, 2007-02-07