计算机科学2000Vol. 27№. 4

并行通信算法的实用性研究*)

Analysis of the Practicality of Parallel Communication Algorithms

陈湘川　王光荣　郑世荣

(中国科学技术大学计算机科学与技术系　合肥230027)

Abstract The significance of communication operations for scalable parallel systems has been well emphasized by the MPI standard. A lot of communication algorithms have been published for efficient communication operations in parallel and distributed systems recently. But a careful reading shows that different models are used and some do not fully consider the characteristic of underlying architecture in performance evaluation. Therefore comparisons are nearly impossible. This makes it difficult to select the best ones for practical implementation. In this paper, we emphasize on such problems. We analyzed some representative algorithms and give out the problems that should be considered to make such kind of algorithms more practical. We also give out some solutions.

Keywords Communication operation, Parallel and distributed system, Performance evaluation

1 引言

众所周知，结点间密集的通信操作会极大地降低并行应用的性能，因而对于并行系统来说，一个有效的通信操作库是极为重要的。常用的通信操作包括：广播，多点播送，路障同步、全对全通信等。

1994年，MPI标准[1]的制定更加突出了并行通信问题的重要性。近年来，针对各种不同的并行系统都有大量的通信算法出现。有些算法的性能分析相当精确，但也有很多算法只是泛泛而谈，内容过于模糊[2,13,15,16]，使算法的性能和应用范围不太清楚，系统设计者很难知道该通信算法是否符合其系统的具体特性和要求。

为了设计出更实用的并行通信算法，本文从通信操作中归结出四种基本动作类型，并从系统、应用和算法三方面对通信算法进行深入探讨，找出了影响算法性能的关键因素，从而提出一些使算法更加实用所需要考虑的问题和解决措施。

2 并行中的通信操作

下面，我们对并行通信所涉及的一些操作做一个简单的介绍，更全面的描述参见文[1]。为这些操作所设计的各种算法参看综述文[13～15,2]。

广播(broadcast)是并行系统中最常见的通信操作，主要用于计算开始前对代码和数据进行分配。这在SPMD编程模式中尤为有用。对于一个基于分布式存储的程序而言，广播可以用于计算阶段将数据从某一个结点发往其他所有结点。而对于使用共享存储或分布式共享存储的应用来说，使用广播操作可以方便地通知其他处理机发生在本地处理机上的事件，如释放一个存储锁等。本质上，广播操作就是不带结点标号(无标号)的定长的数据在系统内的一个单对全操作，而当多个广播同时发生时，就成为一个多对全操作，极端的例子是每一个结点都同时对其他结点做广播操作，这时称为全对全操作。

多点播送(multicast)是广播的一个子集，在MPI标准里没有单独描述。该操作主要表现为无标号的定长数据在结点间的一个单对多传播。同广播一样，它被广泛地用于对代码、数据、事件等进行分配和传播。在分布式共享存储系统中，特别是对于那些支持基于目录的cache密致性(coherency)协议的系统，多点播送常常被作为实现cache写无效/更新的基本手段。在用户级上的多个广播的并发执行，如果参与的进程群/处理机群不相交，则将在系统级上形成多个多点播送。

路障同步常被用于同步并行程序在多个进程/处理机上的执行。在分布式存储和分布式共享存储系统中，它能够很好地支持共享数据上的生产者-消费者关系。在功能上，路障同步表现为一个多对单通信阶段接

*)本文得到国家自然科学基金(69673043号)资助。

一个单对多通信阶段。

全局归约(global reduction)主要用于分布式存储系统。全局归约操作包括 max、min、sum 和任何用户定义的函数。MPI 标准允许两种版本的操作。在第一种中,总的结果只能提供给一个参与者。而第二种版本允许所有的参与者知道。与路障同步相似,全局归约操作同样表现为一个多对单通信阶段接一个单对多通信阶段。当然,第一种版本不需要后面这个阶段。

收集(gather)操作是指一个参与者从某个进程群/处理机群上收集数据的过程。它与路障同步和归约不同的是:它所处理的数据是有标号(进程/结点号)的,在操作执行过程中,传输的数据量是不断增加的。全收集(allgather)是收集操作的一个变形,它向所有的参与者提供收集结果。分散(scatter)操作是收集的逆过程,它将数据从某一个结点分配至其他结点,与广播不同的是传播的数据是有标号的,也就是说:每个参与者从发送者得到的数据是不同的。这类操作主要用于分布式存贮系统中。

全对全通信操作(gossip)结合了收集和分散两个操作。在这个操作里,每个参与者(进程/结点)都拥有一片带标号的信息。操作结束时,每个参与者都知道所有的信息片。一连串的收集和分散操作交织在一起以得到需要的结果。全对全通信主要用于分布式存储系统,并在数据重分配中受到愈来愈高的重视。

通过分析上述并行通信操作内在的要求,我们得出了组成这些操作的四个基本动作类型(见表1)。任何一个可以有效地实现这四个动作的并行系统都能实现快速的通信操作。其中两个强调分配的数据是否带标号,其余两个强调收集的数据的长度是否随着操作的进行而变化。很显然,不定长数据的收集比定长的复杂。从表中可以看出,分配无标号数据和收集定长数据在通信操作中较为普遍。

表1 标识并行通信操作的四种基本动作类型

通信操作	分配无标号数据	收集定长的数据	收集不定长数据	分配有标号数据
广播	是			
多点播送	是			
路障同步	是(第二阶段)	是(第一阶段)		
全局归约(结果通知一个参与者)		是		
全局归约(结果通知所有参与者)	是(第二阶段)	是(第一阶段)		
收集			是	
全收集	是(第二阶段)		是(第一阶段)	
分散				是
全对全通信			是	是

3 算法实用性分析

下面,我们将从系统、应用和算法这三个角度对现有的一些通信算法进行全面的分析,探讨它们所具有的特征、适用范围以及局限性等。

3.1 系统观点

为了使所设计的算法实用,首先要考虑因系统本身的特性不同给算法实用性带来的影响。

3.1.1 报文交换与线路交换 大部分算法是基于报文交换的(也称为存储转发)。在这种模式下,结点只能与相邻结点直接通信。也有一些算法是基于线路交换技术的。这种模式允许两个结点通过一条通路交流信息[16]。而蛀孔(Wormhole)常常被看作线路交换的一种特例。

由于采用线路交换技术的机器都能执行基于报文交换的应用,因此,一些在报文交换模型上性能很好的算法被简单地认为在线路交换模型上也同样优秀。但是,应该看到基于线路交换的算法其通信步的时间长短紧密地依靠于机器本身的参数,比如启动时间、通信链路的带宽、路由的延迟等等。这些都将影响算法在机器上的真正性能。很遗憾的是,大多数算法在性能分析时都没有考虑这些开销。另一方面,大多数基于线路交换的算法所需要的通信步数的确比相应的报文交换算法少一些。但由于同样的原因,很难判断谁更适用。

此外,即使一个算法是专门为线路交换设计的,其适用范围和性能还依赖于它所采用的是点不相交通路模型还是边不相交通路模型。更进一步,还应当指出它所采用的是严格协议[16]还是宽松协议。这些因素都会在相当大的程度上影响算法的性能表现。

因此,当评价一个算法时,我们应该仔细地将以上

问题考虑在内。

3.1.2 单口与多口 所有的算法都是基于某种通信端口模型的，包括单口、全口和多口三种。端口模型说明了系统的输入和输出能力。大多数算法采用最简单的单口模型，一个结点同时只能使用一条链路作为输入/输出通道。因此在一个通信步内，结点只能接收或(和)发送一个消息。另一些算法采用全口模型[4,5]，一个结点可以同时使用所有的链路作为输入/输出通道。还有一些结点采取了权衡策略，以机器的DMA能力作为端口数目的限制，称为d口模型($1 \leqslant d \leqslant n$，n为结点的度)。

绝大多数多端口算法都表明它们所需要的通信步的数目比相应的单口算法少。然而，这里通常存在着一个微妙的假定，即：不管系统是单口还是多口，一个通信步的时间是相同的。对于实际应用中的并行系统来说，这一假定未免太强了。它意味着，通信链路就是在系统缓冲区和路由间进行传输的"DMA通道"。对于多端口接收来说，一旦缺乏足够的存储带宽，相关的输入链路就会争用存储总线并不得不多路复用。同样的情况也会在多口输出操作中出现。此外在异步多口输出操作中，我们还应该注意在系统缓冲区中创建多个消息头和多个数据拷贝的额外费用。

因此，在一个通信步内完成并发接收和发送的定义不再适用，除非将上面所提到的相关系统因素都考虑进去。一个简单的解决方案是：d口系统中一个通信步的时间是单口系统中的d倍。在这种假定下，很难仅仅因为通信步的数目少一点就说多口算法比单口算法好。文[6]对通信步时间模型进行了详细的分析。这种分析是非常必要的。只有这样，才能在多口算法之间，多口与单口算法之间进行公平的比较。这种分析还能帮助我们判断一个多口算法是否适合某个具有特定输入输出能力的给定系统。

3.1.3 确定性路由与适应性路由 现有的并行系统支持的是确定性路由，但在理论上已经有几种适应性路由策略提出，而且将应用于新一代并行系统中。这导致了两个问题：1)如何使在确定性路由上设计的算法在适应性路由上也有同样好的表现；2)如何利用适应性路由的特点来减少通信代价。

如果我们考虑现有的算法，可以肯定有一些是无需改变的。比如一个使用分而治之策略的单源广播(单对全)算法，无论在确定性下还是在适应性路由下都有一样的性能。然而一个多源多点播送(多对多)算法在两种不同的路由策略下将不会有同样的表现。很容易证实，目前现有的部分全对全算法，在适应性路由下将出现在确定性路由下所没有的链路争用。因此，当在采用适应性路由的系统里引入为确定性路由而设计的算法时，我们必须确信它不会导致附加的结点/链路争用，从而使系统性能下降。

第二个问题目前还是一个挑战。为解决这一问题，必须建立新的设计框架，捕捉适应性路由策略中的各种争用问题，最终得到新的性能更好的算法。

3.2 应用观点

3.2.1 动态与静态 一个算法是静态的是指操作开始时每一个参与者就已经知道所有其他参与者或要打交道的那一部分参与者的标志，典型的如结点/进程标号等。以文[7]中所提出的多点播送算法为例。它采用了分而治之的策略。在一个静态实现中，所有的中间目的结点都需要知道各自的传输对象是什么，这样它们才能在收到消息后尽快地将消息送出。然而，对于一个动态实现而言，在结点间传送的消息除了实际的数据外还必须加有一个剩余的结点清单。一个结点在收到消息后，必须将目的地清单与数据分离，计算下一个目的结点，重写目的结点清单，合并数据和新的清单，然后才能将新构成的消息送往下一个目的结点。这不仅会带来额外的计算开销，也会带来额外的消息延迟，因为在中间目的结点上消息的长度会被改变。然而，很少有算法在说明和评价自己时，将这些开销考虑在内。

问题是动态应用并不少见。比如基于目录的cache密致性协议[1]中的写无效/更新操作就必须使用多点播送的动态实现，因为仅仅源结点知道目的结点清单，而目的结点相互间并不知道。因此，当一个多点播送算法声称它适用于cache密致性的时候，它应该将前面提到的额外开销综合进性能评估里。否则，要么很难预测算法动态实现的效率，要么算法的应用范围只能局限于静态实现中。

除了多点播送外，其他的通信操作包括全局归约、路障同步等也存在类似问题。因此，为这些操作设计的算法必须明确其适用范围，并采用合适的评估策略，使其具有更高的实用价值。

3.2.2 单个操作与多个并发操作 许多时候，一个新算法总是依据它在单个操作上的性能来证明自己比其他的算法更好，但在实际系统中，多个同类通信操作并发是比较常见的。因此，在单个操作上的优良性能并不意味着在多个操作并发时比别的算法强。比如一个在单个路障上有上佳性能的算法并不需要在多发路障上也表现优秀，因此，当提出一个通信算法时，应该清楚地指出它是为单个操作还是多个并发操作设计的，同时也需要给出性能分析的依据。

即使一个算法是专门为多个并发操作而设计的，其适用范围和性能评估还取决于各组参与者是否相交。这有可能导致结点争用和链路争用。比如，结点X

同时参与了两个多点播送操作 Y 和 Z，那么一个并发操作的算法应该避免操作 Y 和 Z 在同一步要求 X 发送消息，否则将导致结点争用，操作 Y 和 Z 将在 X 结点上串行，从而延迟两个操作的时间，即使 Y 和 Z 的参与者是不相交的，各自的参与者也可能在并行系统的同一块内，从而导致链路争用。

此外，其他的通信操作如路障同步、收集、全局归约都受到类似问题的困扰。因此，任何为多个并发操作而设计的算法需要清楚地定义和评估自己的适用范围。

3.3 算法观点

现在，让我们从算法本身的角度来分析实用性问题。

3.3.1 *同步与异步* 许多算法都采用同步的方法来解决如全对全通信等[9,10,17]问题。也就是说，所有的参与者同时开始一个通信步，也同时结束该步，在每一步内，是没有资源争用的。这种策略使算法可以很容易得到通信复杂度的上下界。然而，由于内在的同步性，当其在目前主流的 MIMD 机器上实现时，它们的性能并非如所期望的那么好，这使得在实际应用中，很难选择合适的算法。因此，在设计算法时就应该对机器的内在异步性加以考虑。

3.3.2 *直接、间接与混合方法* 一些为全对全通信而设计的算法在本质上就是直接型的，每个结点通过不同的通信步将消息送往每一个目的结点。在间接型的算法[9,12,18]中，结点可以从多个结点接收消息，排列数据，重组消息，然后将它送往另一个结点集。还有一些算法组合了直接和间接两种方式，称为混合方法。除了通信启动时间和消息传播代价外，间接型和混合型算法的性能还受到数据排列转换代价的影响。这种代价还取决于数据是从缓冲区直接传入缓冲区，还是从缓冲区到存储器再到缓冲区。虽然现在的算法考虑了数据转换代价，但缓冲区的数目和用途需要清楚地说明。另一个重要的问题是绝大多数间接型和混合型算法忽略了系统所允许的最大消息长度。这些算法的最优性能是在消息长度不受限的情况下得出的。为了使这类算法实用，在性能分析时应该考虑最大消息量[13]的限制。

3.3.3 *线性时间模型与常量时间模型* 现在，我们考虑可能的通信时间模型。大量的实验表明，两个结点的通信时间极大地依赖于消息长度[11,13]，线性时间模型遵从这一点，一个通信步所需的时间随着消息量的上升而成比例上升，遗憾的是，很多算法为了理论证明的方便，简单地采用了常量时间模型[3]，在这种模型里，一个通信步只需一个时间单位，无论消息的量有多大，传输路径有多长；而且消息的分裂和重组被假定为不需要时间。因此采用常量时间模型的算法通常"具有"更好的性能。在实际应用中，这样的模型几乎不可能在蛀孔算法(wormhole)之外的机器上实现。由于采用常量时间模型的算法难以给出真正的时间复杂度，因此无法与采用线性时间模型的算法进行比较。为了让这类算法变得实用，应该将消息量和路径长度考虑在内。

结论 本文从系统、应用和算法三个角度对并行通信算法的实用性进行了分析研究，探讨了如下几个方面的问题：1)如何确定现存算法的应用范围；2)如何正确全面地分析算法的各种性能；3)如何为当前和未来并行系统设计更实用的算法。同时，本文也给出了一些相应的解决措施。

(参考文献共19篇，略)

(上接第84页)

用 Z 语言来说明，计算视点可以用 LOTOS 语言来说明。可以看出，特定的规范说明语言可以作为特定的视点语言，而且每个语言的描述程度是不同的。这就引出一些问题：对于每个 ODP 视点选择何种 FDT 最好；如何保证不同 FDT 书写的不同视点说明之间的一致性等。

基于视点技术的开放式分布处理是设计分布式多媒体的一个比较好的方法，但现在还有一些亟待解决的问题。目前已有一些学者开始着手解决这些问题。

参考文献

1 ITU Recommendation X.901-904—ISO/IEC 10746 1-4: "Open Distributed Processing — Reference Model — Parts 1-4", July 1995

2 Zimmermann, H. OSI-reference model-ISO model of architecture for open system interconnection. IEEE Trans., 1980, COM-28: 425～432

3 Editoral: Viewpoints in Requirements Engineering. Softw. Eng. J., 1996, 11(1): 2～4

4 Berzsenyi M, et al. Design and implementation of a video on-demand system. Computer Networks and ISDN systems, 1998, 30(16/18): 1467～1473

5 Leydekkers P, Gay V. Multimedia conferencing services in an open distributed environment. IWACA '94

6 Geihs K, Mann A. ODP viewpoints of IBCN service management. Computer Communications, 1993, 16(11): 695～705

7 Bowman H, et al. Cross-viewpoint consistency in open distributed processing. Softw. Eng. J., 1996, 11(1): 44～57

8 Linington P F. RM-ODP the architecture. In: Raymond K, Armstrong L, eds. IFIP TC6 Int. Conf. On Open Distributed Processing. Brisbane, Australia, February 1995. 15～23

9 Geihs K. The Road to Open Distributed Processing (ODP). In: Proc. GI/NTG Conf. "Communication in Distributed Systems. Mannheim, Germany, February 1991