MPI 集合通信剖析技术的研究

崔 奇,谷建华
(西北工业大学 计算机学院,陕西 西安 710072)

摘 要:将 MPI(Message Passing Interface)进程拓扑有效地映射到处理器拓扑上有助于提高 MPI 程序的通信性能。目前大部分的 MPI 进程映射只考虑点对点通信,很少考虑到集合通信,原因是获取集合通信的进程拓扑是比较困难的。目前大部分剖析(profiling)工具在剖析集合通信时只考虑了函数的接口语义,而忽视了实现语义,导致这些工具不能正确地获取集合通信进程之间的详细通信情况。文中提出了一套剖析算法,可以准确地计算出参与集合通信的每对进程之间的通信量,并以通信矩阵的形式给出进程拓扑。实验证明了剖析算法的正确性,并且通过这种剖析方法获取的进程拓扑能够提升进程到处理器核的映射实验效果。

关键词:MPI;集合通信;通信剖析;进程映射
中图分类号:TP319 **文献标识码**:A **文章编号**:1673-629X(2013)10-0031-05
doi:10.3969/j.issn.1673-629X.2013.10.008

Research of Collective Communications Profiling in MPI

CUI Qi,GU Jian-hua
(College of Computer,Northwestern Polytechnical University,Xi'an 710072,China)

Abstract:The mapping from processes to processors can help the Message Passing Interface (MPI) application improve the communication performance of parallel program. Most of the mappings at present are based on point-to-point communications,and barely consider the collective communications,because it is difficult to obtain the process topology of collective communications. Most of the profiling tools nowadays only think over the interface semantics of functions in analyzing collective communications,and ignore the implementation semantics,leading to these tools can't acquire detailed communication situation of processes accurately in collective communications. Present a set of profiling algorithm which can obtain the communication traffic for each pair of processes participating collective communications accurately. The experimental results show the correctness of the profiling method,and the process topology acquired through this profiling method can get better effect in the mapping from process to core.

Key words:MPI;collective communication;communication profiling;process mapping

0 引 言

在现代集群中通信性能能够根据进程在处理器上的位置的变化而显著改变。在系统中,进程和数据距离越近,访问数据的速度就越快,进程间的通信表现为非一致性,这种情况在集群系统中是非常常见的。为了解决这个问题,一些研究者提出了一些方法,有些是从 MPI 的编程方式和并行算法入手的[1],还有一些研究者提出 MPI 进程映射的方法[2-4],具体做法是将 MPI 的进程拓扑映射到底层的硬件拓扑上,即要求通信亲和度越大的进程应该被放在一些通信代价越小的处理器上。通过进程映射能够使 MPI 程序充分挖掘硬件体系结构的潜能,提高程序的通信性能。而 MPI 进程映射的第一步就是要获取 MPI 程序的进程拓扑[5]。

要获取 MPI 程序的通信拓扑,必须先记录下来 MPI 程序中每对进程之间的通信情况,比如说通信量或通信消息数量,这可以通过对 MPI 程序的剖析来实现[5]。但是常用的剖析工具,比如 mpiP,VampirTrace,MPE 等只能获取点对点通信的通信情况,而对于集合通信,这些工具无法正确地获取参与通信的每对进程之间的详细通信情况[6]。这主要是因为这些工具没有正确区分 MPI 函数的接口语义和实现语义。经过分析,文中认为 MPI 函数的接口语义和实现语义是不同的。接口语义指的是由集合通信函数的功能决定的一

收稿日期:2012-12-19 **修回日期**:2013-03-25 **网络出版时间**:2013-07-24
基金项目:西北工业大学研究生创业种子基金(z2012140)
作者简介:崔 奇(1987-),男,硕士研究生,研究方向为计算机应用技术;谷建华,教授,研究方向为分布式计算、高性能计算。
网络出版地址:http://www.cnki.net/kcms/detail/61.1450.TP.20130724.0953.022.html

种直观上的功能描述。实现语义指的是由函数的功能和函数的内部实现算法共同决定的含义,通过该含义用户能够了解到该函数的实现和语义细节。一般地,在 MPI 的实现版本中集合通信函数的接口语义和实现语义是不同的,而且实现语义要比接口语义复杂得多,实现语义还能反映参与集合通信的进程的真实通信情况。通常的剖析工具都只能获取集合通信接口语义上的通信量,而无法获得实现语义上的通信量。例如对于广播操作,使用剖析工具只能获取根进程广播的数据量,并不能获知在一次广播通信中每对进程之间真实的通信量。

文中针对一种比较流行的 MPI 实现 MPICH2,提出了一种基于实现的集合通信剖析方法,该方法能够准确地获得集合通信的实现语义上的通信数据,并且以通信矩阵的形式给出剖析结果。

1 MPICH2 中集合通信的实现

MPICH2 中集合通信是由一系列 ADI(Abstract Device Interface)[7-9]层的点对点通信实现的,每个集合通信函数根据其运行时参数选用不同的算法,以达到最优的性能。MPICH2 的集合通信中有几种比较典型的算法[10]:二项树算法(binomial tree algorithm)、递归加倍算法(recursive doubling algorithm)和环形算法(ring algorithm)等。大部分的集合通信函数都是由这些算法组合而成的。此节以 MPI_Bcast 为例描述这些算法的原理。

1.1 MPI_Bcast 中的二项树算法

二项树广播算法用于广播小消息,算法的基本思想如图 1 所示。第一步,根进程发送数据给进程(root + $p/2$),root 表示根进程号,p 表示进程个数;该进程收到数据之后和根进程一样把自己作为所在的子树中的新的根进程,然后递归地调用这个算法直到所有的进程都收到广播数据。

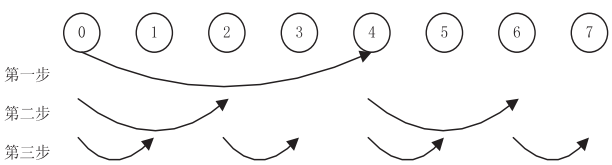

图 1 二项树广播算法

1.2 MPI_Bcast 中的递归加倍算法

对于中等消息和大消息,根进程先采用 scatter 操作将消息的不同部分分散给不同的进程,然后使用一个 allgather 操作将数据收回。对于中等消息并且是进程数为 2 的整数次幂的情况,allgather 操作采用递归加倍算法,图 2 描述了递归加倍算法。第一步,距离相隔为 1 的进程交换它们拥有的数据片段;第二步,距离相隔为 2 的进程交换它们拥有的数据片段和之前收到的数据片段。以此类推,直到所有进程都获得了全部的数据片段,将这些数据片段组合起来就构成了根进程所广播的消息。

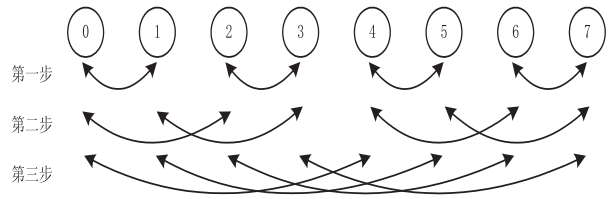

图 2 递归加倍算法

1.3 MPI_Bcast 中的环形算法

当要广播的消息是大消息或者是中等消息但进程个数非 2 的整数次幂的时候,allgather 操作采用的算法是环状算法。图 3 描述了环状算法的 allgather 操作的基本原理,每个进程都向其右邻居发送数据,从其左邻居接收数据,直到所有的数据片段都在环上传递一圈。

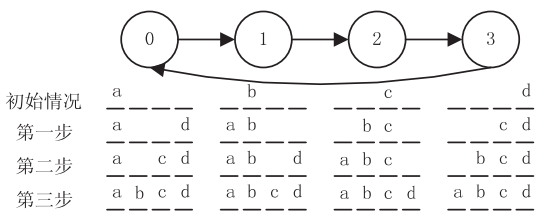

图 3 环状算法

2 分析与设计

2.1 分析和原理

从第一节中可以看出集合通信的实现语义和它们的接口语义有很大的不同,因此,按照这两种语义获取的广播通信的通信矩阵必定是不同的。例如:假设有 8 个进程,现在 0 号进程要广播一个大小为 10 个字节的消息,如果按照广播操作的接口语义,这 8 个进程的通信矩阵为图 4(a),但是根据广播操作的实现语义,这时采用的是二项树广播算法,实际的通信矩阵为图 4(b)。从图 4 中可以看出这两种情况的通信矩阵的差异是比较大的。

但是直接获取集合通信实现语义上的通信矩阵是比较困难的。目前的剖析工具一般是利用 MPI 本身提供的剖析接口(MPI Profiling Interface)对 MPI 程序进行剖析的,而这些接口只能获取 MPI 函数的接口语义信息,无法提供任何有关集合通信的实现语义信息,因此就无法通过这些剖析接口直接获得集合通信的真实通信矩阵。

文中通过分析 MPICH2-1.4.1 中的集合通信源码,获取其内部的实现细节,然后针对这些集合通信的内部实现设计出一套剖析算法,这些剖析算法能够根据 MPI 程序中集合通信函数调用时的输入参数,准确

proc	0	1	2	3	4	5	6	7
0	0	10	10	10	10	10	10	10
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0

(a)

proc	0	1	2	3	4	5	6	7
0	0	10	10	0	10	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	10	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	10	10	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	10
7	0	0	0	0	0	0	0	0

(b)

图 4　MPI_Bcast 操作的通信矩阵

地计算出在集合通信过程中每对进程之间的通信量,从中获取集合通信的通信矩阵。

剖析算法的输入是 MPI 集合通信的运行时参数,包括:进程个数、消息大小、进程号、根进程号等。剖析算法的输出是一个通信向量,向量的长度和进程个数相等,向量中的每个元素表示在此次集合通信过程中该进程向其他进程发送的消息总字节数。

2.2　集合通信剖析算法

二项树算法和递归加倍算法是 MPICH2 中集合通信函数最常用的两种算法,文中将以这两种算法为例介绍对应的剖析算法。

算法 1 是针对二项树算法的剖析算法的伪代码,输入参数为消息大小和进程个数,为了方便叙述,在算法中假设 0 号进程为根进程。剖析算法首先计算该进程第一次发送数据时它的子树的长度,子树的长度表示该子树中进程的个数,然后计算出该进程每次发送数据的目的进程号以及发送的字节数。

算法 1:针对二项树算法的剖析算法。

输入参数:消息大小(msg_size)、进程号(rank)、进程个数(comm_size)。

输出参数:通信向量(comm_volume_vector)。

```
1. pof2← 1
2. while pof2 < comm_size do
3. if rank & pof2 == 1 then
4. break
5. end if
6. pof2← pof2 << 1
7. end while
8. length_of_subtree← pof2
9. pof2← 1
10. while pof2 < length_of_subtree do
11. receiver← rank + pof2//计算目的进程号
12. if receiver < comm_size do
13. comm_volume_vector[receiver]← msg_size
14. end if
15. pof2← pof2 << 1
16. end while
```

算法 2 是针对递归加倍算法的剖析算法。如果进程个数为 p,则剖析算法会执行 $\log_2 p$ 步。第 i 步执行的操作是为每一对相隔距离为 2^{i-1} 的进程计算通信的目的进程和通信量。

算法 2:针对递归加倍算法的剖析算法。

输入参数:消息大小(msg_size)、进程号(rank)、进程个数(comm_size)。

输出参数:通信向量(comm_volume_vector)。

```
1. 该进程当前拥有的数据量 current_size ← msg_size
2. 两个交换数据的进程之间的距离 pof2 ← 1
3. while pof2 < comm_size do
4. des← rank ^ pof2 //计算目的进程号
5. if des < comm_size then
6. comm_volume_vector[des]← current_size
7. current_size← current_size * 2 //该进程拥有的数据量加倍
8. end if
9. pof2← pof2 << 1
10. end while
```

2.3　数据收集

为了获得整个 MPI 程序的通信矩阵,Profiling 程序会在 MPI 程序的初始化阶段为每个进程创建一个全局的通信向量,用来记录该进程发送给其他进程的总字节数。在对 MPI 程序剖析的过程中,遇到点对点通信,Profiling 程序会记录下目的进程号和通信字节数,直接生成通信向量。如果遇到集合通信则会通过集合通信剖析算法计算通信向量。每次获得通信向量后都将其累加到该进程的全局通信向量中。

在 MPI 程序运行结束时,Profiling 程序中的数据收集进程(一般是 0 号进程)会从其他进程中将它们的通信向量收集起来,按照进程号从小到大的顺序组成通信矩阵,原来每个进程的通信向量都会成为通信矩阵中的一行。文中使用通信矩阵来表示进程拓扑。

3 实验评估与分析

3.1 实验设置

为了验证文中所提出的集合通信剖析算法的正确性,在 MPICH2-1.4.1 的源码中的 ADI 模块中加入了一些探针代码,用来获取几个基本的 ADI 层的点对点通信函数的通信情况。由于集合通信的内部实现使用的点对点通信函数都来自于 ADI 层,所以在这一层获取的数据可以真实地反映出整个 MPI 程序的通信情况。文中用集合通信剖析算法计算出的通信矩阵和探针代码在 ADI 层获取的通信矩阵进行比较来验证剖析算法的正确性。

为了说明集合通信剖析算法对映射实验效果的影响,实验重点比较了考虑了集合通信的进程映射效果和没有考虑集合通信的进程映射效果,映射算法采用 TreeMatch[3] 算法。文中在映射试验中进行了两类性能评测实验:第 1 类属于微基准(micro benchmark)测试,目的在于评价单一集合通信在考虑了实现语义时的映射效果,在这类试验中采用的微基准测试程序为 Intel MPI Benchmarks(IMB),版本为 3.2.2;第 2 类实验是用一些复杂的应用来进行性能评测,目的在于评价在复杂场景下,考虑了集合通信的映射效果,在这类试验中采用了 NAS Parallel Benchmark(NPB),版本为 3.3。实验使用 MPICH2 的进程管理器 hydra 提供的绑定机制来实现进程到核的绑定。

文中的实验环境是一个拥有 8 个节点的曙光集群系统,每个节点内有两颗 2.5 GHz 的 AMD Opteron 2380 四核处理器和 8 GB 内存,每个核拥有 2 M 的二级缓存,每个处理器上的四个核共享 6 M 的三级缓存。节点间用千兆以太网互联。实验平台的操作系统使用 Ubuntu Enterprise Linux 11.10,MPI 库采用 MPICH2-1.4.1,编译器使用的是 GNU 编译器,版本为 4.6.1。

为了叙述方便,在下面的小节中将考虑了集合通信实现语义的剖析方法叫做基于实现的剖析方法,将没有考虑集合通信的实现语义的剖析方法叫做传统剖析方法,将使用了基于实现的剖析方法的进程映射叫做优化映射,将使用了传统剖析方法的进程映射叫做传统映射。

3.2 实验结果与分析

实验发现剖析算法计算得到的通信矩阵与探针代码在 ADI 层获取的通信矩阵完全一致,可以说明文中的集合通信剖析算法是正确的,它能够真实地反映 MPI 程序的通信特征。下面是两类映射实验的结果和分析。

3.2.1 微基准测试

微基准测试的目的是验证在单个集合通信的进程映射中使用剖析算法获取的进程拓扑对于提高映射效果是否有帮助。实验评价指标是加速比:

$$\text{Speedup} = T_{\text{default}} / T_{\text{opt}}$$

T_{default} 为使用 MPICH2 默认的映射方式的运行时间;T_{opt} 是按照优化映射后的运行时间。实验考虑了通信消息的大小和进程数是否为 2 的整数次幂这两种情况对映射效果的影响。

图 5 给出了 MPI_Bcast 和 MPI_Allgather 的优化映射效果。在图中横轴表示消息大小,纵轴表示加速比。从这些图中可以看出针对单个集合通信进行进程映射是有效果的,加速比一般都大 1,这说明了基于实现的剖析方法获取的实现语义上的进程拓扑在单个的集合通信的映射中起了显著的效果。

从图 5 中可以看出,进程个数为非 2 的整数次幂的情况对集合通信的映射效果是有负面影响的。MPI_Bcast 和 MPI_Allgather 在进程个数为非 2 的整数次幂的时候使用的是环状算法或类似环状的算法,而使用这个算法时,在映射前后进程的排布是几乎没有什么变化的,所以映射效果不如进程个数为 2 的整数次幂的情况。

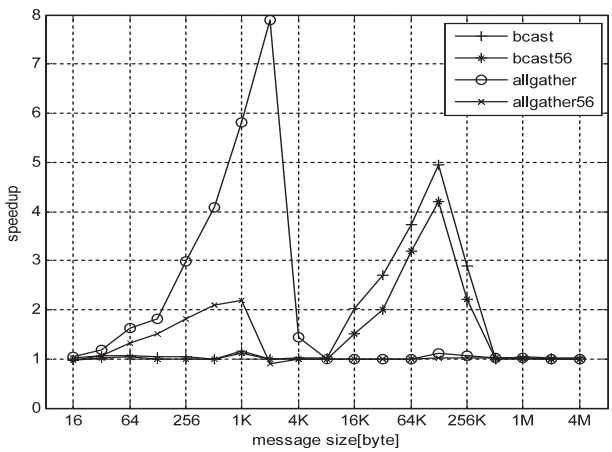

图 5 MPI_Bcast 和 MPI_Allgather 的映射效果(NP=64,NP=56)

3.2.2 复杂应用测试

在复杂应用测试中文中使用了 NPB 中的 8 个基准测试程序进行测试,文中的实验选择了 NPB 中的 C 级程序,进程数都为 64 个。此类实验的目的在于验证优化映射效果是否比传统映射的效果显著,衡量指标是两种映射的加速比(见表 1)。

从表 1 中可以看出,除了 FT 和 IS 两个程序之外,优化映射的加速比相对于传统映射的加速比都有所提高。FT 和 IS 加速比没有提高的原因是这两个程序大量使用了 Alltoall 和 Alltoallv 这两个集合通信函数,而这两个集合通信是多对多类型的通信,从获得的通信矩阵中可以看出每对进程之间的通信都很频繁而且通信量基本相等,这对于启发式映射算法来说是无法找到一个比较好的映射布局。实验还统计了 NPB 中集

合通信函数的调用次数占所有 MPI 通信函数调用次数的百分比,发现集合通信函数动态使用的比例越大优化映射效果就越好。

表 1　NPB 映射效果

程序名称	默认映射时间/s	传统映射		优化映射	
		时间/s	加速比	时间/s	加速比
bt. C. 64	77.08	75.48	1.02	73.03	1.06
cg. C. 64	49.66	46.39	1.07	45.29	1.10
ep. C. 64	9.76	9.66	1.01	9.46	1.03
ft. C. 64	78.36	76.01	1.03	75.84	1.03
is. C. 64	7.56	7.41	1.02	7.39	1.02
lu. C. 64	52.25	46.95	1.11	46.01	1.14
mg. C. 64	8.03	7.41	1.08	6.68	1.20
sp. C. 64	109.23	100.86	1.08	96.84	1.13

4　结束语

文中提出了一套获取针对集合通信的剖析算法,这套算法是根据集合通信函数的实现细节而设计的,通过这套算法能够获取集合通信的通信矩阵。文中通过在 MPI 源码库中加入的探针代码获取的数据证明了剖析算法的正确性。然后通过映射实验证明了剖析算法在实际应用中的有效性。映射实验分为两类:微基准测试和复杂应用测试。映射试验结果表明集合通信的剖析算法对于映射效果的提高作用显著。此外,并行程序中集合通信函数所使用的比例越高,考虑了集合通信的映射效果就越好。

参考文献:

[1] 李　明,张玉敏,SMP 系统上两种并行机制的比较[J]. 计算机工程与科学,1996,18(3):9-15.

[2] Rashti M J,Green J,Balaji P,et al. Multi-core and network aware MPI topology functions[C]//Proceedings of the 18th European MPI Users' Group Conference on Recent Advances in the Message Passing Interface. Heidelberg:[s. n.],2011:50-60.

[3] Jeannot E,Mercier G. Near-optimal placement of MPI processes on hierarchical NUMA architectures[C]//Proceedings of the 16th International Euro-Par Conference on Parallel Processing. Heidelberg:[s. n.],2010:199-210.

[4] 卢兴敬,商　磊,陈　莉. POM:一个 MPI 程序的进程优化映射工具[J]. 计算机工程与科学,2009,31(A1):201-205.

[5] Xu Q,Subhlok J,Zheng R,et al. Logicalization of communication traces from parallel execution[C]//Proc of IEEE International Symposium on Workload Characterization. Houston:[s. n.],2009:34-43.

[6] Mercier G,Clet-Ortega J. Towards an efficient process placement policy for MPI applications in multicore environments[C]//Proceedings of the 16th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface. Heidelberg:[s. n.],2009:104-115.

[7] Gropp W,Lusk E,Doss N,et al. A high-performance,portable implementation of the MPI message passing interface standard[J]. Parallel Computing,1996,22(6):789-828.

[8] Ashton D,Gropp W,Thakur R,et al. The CH3 design for a simple implementation of ADI-3 for MPICH with a TCP-based implementation[R/OL]. 2003. http://phase. hpcc. jp/mirrors/mpi/mpich2/docs/tcpadi3. pdf.

[9] 孙亦嘉,张　岳,陈　渝. 基于 VIA 的 MPICH2 研究与实现[J]. 计算机工程与应用,2005,41(1):98-101.

[10] Thakur R,Gropp W D. Improving the performance of collective operations in MPICH[M]//Recent Advances in Parallel Virtual Machine and Message Passing Interface. Heidelberg:[s. n.],2003:257-267.

(上接第 30 页)

[5] Bulusu N,Heidemann J,Estrin D. GPS-less low-lost outdoor localization for very small devices[J]. IEEE Personal Communications,2000,7(5):28-34.

[6] Blumenthal J,Grossmann R,Golatowshi F,et al. Weighted centroid localization in Zigbee-based sensor networks[C]//Proc of IEEE International Symposium on Intelligent Signal Processing. Alcala de Henares:IEEE,2007:1-6.

[7] Sichitiu M L,Ramadurai V. Localization of wireless sensor networks with a mobile beacon[C]//Proceedings of IEEE MASS. Philadelphia,PA:[s. n.],2004:174-183.

[8] Ssu K F,Ou C H,Jiau H C. Localization with mobile anchor points in wireless sensor networks[J]. IEEE Transactions on Vehicular Technology,2005,54(3):1187-1197.

[9] Koutsonikolas D,Das S M,Hu Y C. Path planning of mobile landmarks for localization in wireless sensor networks[J]. Computer Communications,2007,30(13):2577-2592.

[10] Guo Z,Guo Y,Hong F,et al. Perpendicular intersection:locating wireless sensors with mobile beacon[J]. IEEE Trans on Vehicular Technology,2010,59(7):3501-3509.

[11] Xu Y,Tong C. Design and attitude control of a flying anchor node[C]//Proc of 2011 International Conference on Electronics,Communications and Control. Zhejiang:IEEE,2011:1975-1978.

MPI集合通信剖析技术的研究

作者： 崔奇， 谷建华， CUI Qi， GU Jian-hua
作者单位： 西北工业大学 计算机学院,陕西 西安,710072
刊名： 计算机技术与发展
ISTIC
英文刊名： Computer Technology and Development
年，卷(期)： 2013(10)