MPI设计结构的分析与比较*)

张 岳 陈 渝 孙亦嘉 都志辉
(清华大学计算机科学与技术系 高性能计算研究所 北京100084)

摘 要 MPICH与LAM是目前使用最广泛的MPI标准的实现。本文从设计思想和程序结构方面分析二者实现上的异同,重点比较可移植性和性能,并对二者最新版本进行了详细的性能测试。通过实例分析,提出设备层的设计要点,并为选择、移植和改进MPICH和LAM提出建议。

关键词 MPI实现,MPICH,LAM,性能比较

A Comparison between Design Structures of MPI Implementations

ZHANG Yue CHEN Yu SUN Yi-Jia DU Zhi-Hui
(Department of Computer Science, Tsinghua University, Beijing100084)

Abstract MPICH and LAM are the most comprehensive MPI implementations recently. This paper compares their difference in implementation, porting property and performance according to the design rationale and structure. It also gives guidance for choosing, porting and improving them, by providing analysis in implementation and performance testing results.

Keywords MPI implementation, MPICH, LAM, Performance comparison

在大规模机群系统中,并行程序作为相互独立的进程运行在多个计算节点上。为了实现这样的并行计算,消息传递成为一种有效的方式。使用标准的通信协议—MPI,可以减少设计并行程序的复杂度,提高并行程序的可移植性,保证并行程序的执行性能。早在20世纪90年代,MPI论坛先后公布了MPI标准1和标准2[1,2]。MPI标准是总结以往的消息传递机制的优点而提出的可移植的消息传递标准。MPI标准提供了并行程序通信所使用的API,其中主要有点对点、组通信的调用,多种语言的支持以及差错控制,在MPI2标准中增加了RDMA方式的主动通信和动态进程控制的API调用。目前使用广泛的MPI实现主要有MPICH与LAM,本文将从代码分析和测试入手,对二者进行具体比较,并对可移植性和性能对比做出总结。文章最后对MPI实现的发展进行了展望。

1. MPICH与LAM设计思想的比较

1.1 MPI实现结构

MPI实现主要完成两大工作:(1)通过系统服务管理不同结点进程;(2)MPI库接口实现。第(1)部分工作主要体现在mpirun程序的设计上。以下称第(2)部分工作为MPI库,是本文研究的重点。

1.2 MPI库总体结构

总的来看,MPI库向上给用户提供MPI标准接口,向下利用通信环境实现这些接口。于是,很自然地,MPI库的设计可以分为两层。上层与环境无关,它提供用户接口,并处理与环境无关的MPI结构。而下层与环境有关,实现通信以及通信所必需的消息缓存等功能。为达到可移植的目的,目前的MPI实现大都分为这样两层。以下称上层为API层,而称下层为设备层。

API层的设计对性能是有影响的,比如用户可见结构的存储方式。如果用指针索引它们,那么在参数检验等流程中可能需要多次内存存取操作。但是如果用句柄索引这些结构,并把最重要的信息内置于它们的句柄当中,就可能减少存取开销。

设备层的设计对性能的影响更大。一方面,设备环境的效率千差万别。比如,MVICH[13]是MPICH在VIA[12]环境上移植的版本,由于VIA通信的软件开销比TCP要小得多,因此在机群运行时MVICH的性能要好很多。另一方面,设备层通信算法的设计也起着相当的作用。由于移植工作主要需要重写设备层的代码,所以设备层的实现对MPI库的执行性能尤为重要。

在讨论设备层之前,首先需要具体说明一下API层与设备层的分界点。MPI1标准规定的接口,主要有语言绑定、进程结构、点点通信、组通信以及相关结构控制。其中语言绑定、进程结构可以被API层包办,它们是可移植的部分。组通信可以用点对点的通信实现(实际MPI实现中全都是这样做的),所以也可以移植。于是,点对点的通信及相关数据结构成为API层与设备层交互的界线。这些数据结构的定义,以及放在哪一层实现,对性能和可移植度有着十分重要的影响。

1.3 MPI实现的设备层

1.3.1 设备层核心内容 请求队列,消息缓存和通信协议是设备层设计中最重要的内容,这三者相对独立,但又相互联系,须综合考虑。

请求队列:请求队列的元素是收发请求,它对应于用户可见的MPI_Request。总结请求队列的来源,有以下情况。

1)来自MPI非阻塞发送调用的发送请求对象。为保证公平,这个队列应该是先进先出队列。

2)来自MPI非阻塞接收调用的接收请求对象。它保存为一条有序队列,以便消息到来时查找相匹配的请求。

*)基金项目:本文受2003年自然科学基金资助,基金编号:60203024.张 岳 本科生,主要研究方向为高性能并行通信;陈 渝 讲师,主要研究方向为高性能并行通信,并行编译,操作系统;孙亦嘉 硕士研究生,主要研究方向为高性能并行通信;都志辉 副教授,主要研究方向为并行算法,高性能通信,网格计算。

3)来自物理设备的接收请求对象。当消息已到达,而接收请求队列中还没有相应的 MPI_Request,则到来的消息以一定的结构存放在队列中,以便有了接收调用之后再判断是否匹配。显见,这条队列也应作有序队列组织。

由于阻塞通信一般由非阻塞通信完成,因此请求队列属于设备层的核心结构。

MPICH 和 LAM 处理非阻塞通信的一般过程是:建立一个收发请求(MPI_Request,用户进程可见)对象,然后把这个对象放入相应队列。此时通信调用返回,队列处理与后面的计算同步进行。等队列中的相关收发对象处理完成以后,便从队列中取下这个结构,回收或者重用。而真正的发送接收过程,是在队列处理函数中完成的。

消息缓存:用户要发的消息,有时需要在设备层创建一份拷贝。缓存消息应该尽量少,因为数据拷贝是不小的开销。但在以下的三种情况下,消息缓存是必要的:

1)用户定义的数据是非连续的,而物理设备只接收连续消息;

2)物理设备收到了信息,但用户并没有处理,这时应该在设备层把它们缓存起来,以备后来之需。

3)用户进行了非阻塞的通信,然后返回。用户可能去改变用户区的消息,如不缓存则可能导致不一致。

针对以上这三种情况,可以制定相应的避免缓存手段。对于情况(1),如果想避免拷贝数据带来的开销,就需要定义一种发送方式(或在一定的硬件支持下),把非连续的信息分段发送。对于情况(2),可以利用握手等通信协议涉及技巧,避免意外消息的到来。对于情况(3),MPI 标准制定了缓存与不缓存两种方式,控制用户行为。总之,缓存问题须综合地考虑,设计尽量满足各种情况的方案。

发送接收:不同系统上 MPI 库的本质区别之一就是发送接收的环境不同。处理请求队列的关键步骤,就是依据具体环境提供的通信接口实现通信。通信的协议由设备层设计者来决定,这也是提高性能的重要手段。同是 tcp 通信,MPICH 与 LAM 在对待不同长短的信息时采用不同的策略,进行了不同侧面的优化,得到不同的性能结果(具体见后面性能分析)。

而针对高性能用户层通讯协议 Virtual Interface Architectu-re(VIA)的 MPI 设备层实现,则需要采用与 TCP 不同的设计思路。在 VIA 协议规定,物理设备动作之前必须获得描述子,如果没有描述子,设备会丢弃接收到的信息。为了避免丢包和提高效率,必须要在 MPI 中实现流控机制。如 MVICH[13],设备层采用一种"credit"的流控机制和接收描述子预留机制,避免了丢包的发生,提高了传输的性能。

1.3.2 MPICH 的设备层 MPICH 的设备层叫做 ADI 层[8]。在设计 MPICH 的时候,William Gropp 和 Ewing Lusk 提出了 ADI(Abstract Device Interface)。ADI 提出的目的是为 MPI 等上层应用,提供一种与具体设备无关的通信接口。ADI 层的接口都以 MPID_开头。相对 ADI 层,MPICH 的 API 层函数以 MPIR_开头。

MPICH 把请求队列按类型组织成两种。发送请求用一条队列存放,叫做 MPIR_Sendq。两种接收请求共同组织起来,叫 MPID_recvs。MPICH 对队列的处理完全放在设备层中,这给了 ADI 层相当的独立性。实际上,ADI 可以作为一套完整的点通信调用的接口。在文[8]中可以看到相关的程序。这样,重写时不必考虑与 API 层的联系,所以相对要自由许多。ADI 用的队列处理函数是 MPID_DeviceCheck,它负责对两条队列中的请求进行处理。

通信协议方面,MPICH 采用三种方式。短消息用 short 协议通信,长消息以 eager 协议通信,而超长消息用 rendezvous 协议通信。发送短消息时,ADI 把消息体放入消息信封一起发送。增加了一次拷贝,但却省去了单独发送消息体的开销。发送长消息时的 eager 协议指,发送方发出消息时不必考虑接收方状态。接收下来的消息被缓存在接收设备中,这需要接收端有足够的缓存。在接收时,需要从缓冲把数据拷贝到用户区。而发送超长消息时用 rendezvous,这种协议在发送消息之前先发送一个请求包,等接收端有了回应之后再发送消息。这样做接收端不必缓存消息,而是准备好接收目的以后再给回应,由于该协议避免了大量的数据拷贝,因此 ADI 在长消息上的协议优化是有效的。

MPICH 不将所有长消息都用 rendezvous 协议,是因为握手包占用了带宽。把这部分带宽与数据拷贝的代价向权衡,可得到 eager 与 rendezvous 的分界线。

ADI 对多设备的支持

_MPID_Device 结构是 MPICH 的关键部分。ADI 把每个通信结点看作一个 Device,在全局设置抽象设备链。这样做,是为了强调设备的独立性。对代码进行适当修改,就可以按如下方式支持多设备。在 MPI_Init 执行过程中,把不同设备的函数指针连接到具体通信函数上。而在执行 MPID_开头的函数时,从相应的抽象设备找到连接的函数,并加以调用。

使用抽象设备,多种设备(如本地共享内存和以太网)可以同时运行。但与多设备的好处相伴的是效率的损失。由于设备相互独立,因此在队列处理时也是每个设备分别处理队列。MPI 的队列处理函数为 MPID_Device_Check,从一条请求队列找一个请求时,要让每个设备都去对这条队列搜索一遍。这样做的效率比直接搜索一遍的效率要小很多。

ADI 对可移植性的支持

ADI 对移植和改进和支持是分层次的。在 ADI 最下层,有几个函数,它们是与环境相关的最小集。其中有收发控制消息的 MPID_SendControl,MPID_RecvAnyControl,MPID_ControlMsgAvail,和收发通信消息本身的 MPID_SendChannel,MPID_RecvFromChannel。这五个函数合起来称为 Channel Interface。在最简单的情况下,移植工作就是实现这五个函数。然而,在许多其它情况下,由于性能要求或协议要求等因素,移植时需要重写整个 ADI 层。比如 MVICH,就是重写了全部 ADI 接口。这里很大一部分原因在于,VIA 通信不只依赖于 Send/Recv 原语,还需要用户掌握流控。

1.3.3 LAM 的设备层 LAM 的设计比 MPICH 要简明一些,它以通讯请求为核心。LAM 的设备层叫 Request Progress Interface(RPI),它有两个版本。一个叫 LAMD_RPI,它通过牺牲一定的性能,换取更好的容错性和本身的可移植性。另一个叫 c2c_rpi,这些设备层函数以 mpi_req 开头。因为这样做效率更高,LAM 的设计者推荐通过重写这套接口来移植 LAM。

LAM 中的收发请求用 struct req 结构存放,而结点用 struct process 结构存放,后者非常简单。LAM 把全部通信请求组织在一条队列中,用全局 lam_rq_top 查找。LAM 的队列处理函数为 mpi_req_advanc,它是 RPI 接口的一员。直接从通讯请求出发的思想,使 LAM 将设备层的界线划在了处理请求队列的几个函数上。LAM 的 c2c_rpi 设备层函数正是收发请求操作集合,直接做请求对象的建立与回收,放入与移出队列,以及队列处理。在 LAM 中,请求队列跨于 API 层和设备层之间,这给移植添了麻烦。LAM 的 RPI 层不能像 ADI 层一样,作为一个独立的点对点通信库。

通信协议方面,LAM 原则上只提出两种协议。一种是短

消息协议，不用接收方返回确认信息。与之相对的是长消息协议，需要分段发送，接收方返回确认信息。这样区分长度，主要是为通信缓冲区考虑。

LAM 的速度优势

LAM 的结构特点，使它在效率上占有一定优势。

1)RPI 层的队列处理比 ADI 层要快。RPI 层的队列操作统一，效率较高。由于 ADI 设备有独立性，因此操作全局队列以设备为单位，不如 RPI 快。此外，RPI 层以全局的_req 指针 lam_rq_top 组织请求队列。而 ADI 队列却进行了许多层封装。在频繁使用的结构中，这样做势必影响效率。

2)RPI 层的快速收发大大提高它的速度。当请求队列为空，且目的地址不冲突时，LAM 会避开请求队列，直接进行发送。这个函数叫 rpi_c2c_fastsend。当同样条件满足之时，接收函数也会避开请求队列，直接等待设备信息。由于省去了请求对象的创建于操作，因此可以大幅提升效率。实验证明，这两个函数非常有效。

3)由于 RPI 层对设备的独立性没有要求，因此它可以同时处理多个设备上的收发请求。比如有多个发送请求在请求队列中时，RPI 会进行合理的优化，让多个消息可以共同处理。

2. MPICH 和 LAM 的可移植性比较

通过 MPICH 与 LAM 的具体比较和分析，可以看出 MPICH 在移植方面的几大优势：

1)MPICH 的 ADI 接口清晰。ADI 层的接口是一套独立的点对点通信接口，实现这套接口时不需要许多与 API 的联系。而 RPI 层接口的实现，却依赖请求队列这个结构。此外，ADI 层本身有很大的灵活性。适当情况下，只要实现五个 Channel 接口就可以移植 ADI。

2)MPICH 的设计文档详细。有关移植 MPICH 的文档可见文[8～10]等，而 LAM 的移植文档只有文[11]等几篇大体思路提供。在做移植工作的时候，设计者需要在研读其源码上花费的工夫要小一些。

3)MPICH 的总体结构安排合理。MPICH 的被移植代码在 mpid 目录下，相对比较集中。而 ADI 对移植与改进的支持也留有许多余地。LAM 除了 rpi 目录下的代码要做修改以外，还可能涉及其它代码，调试工作较难。

3. MPICH 和 LAM 的性能评测

两款软件在实际运行中的表现，是选择比较中被普遍关心的事情。从代码分析中可以看到，LAM 的整体性能应该好于 MPICH。但 MPICH 在大消息方面也有自己的优势。对于旧版本的 MPICH 和 LAM，已有过不少评测报告。它们的结论基本与分析一致，但具体结果根据环境、设置而有所不同，可以参见文[5,6]。我们是以目前两个最新版本 MPICH1.2.4 和 LAM6.5.9进行评测。

3.1 运行环境

这次测试使用 Pallas MPI Benchmarks[7]评价，它是一个使用广泛的 MPI 性能测试软件包。集群环境是一个8节点的 cluster，每个节点的配置为：双 AMD AthlonMP 1900＋CPU，512M SDRAM，66MHz/64bit PCI 总线。操作系统是 Linux kernel2.4.18，用100M Ethernet 交换机连接。

3.2 结果比较

在测试的多组函数中，选择有代表性的 PingPong，SendRecv 和 Bcast 进行比较。图中纵轴使用对数刻度，单位是字节。而横轴使用2^n刻度表示，单位为微秒。

PingPong 模式测试　两个进程通信，一方先发送，再接收，另一方先接收，再发送，其余进程阻塞，结果如图1所示。包长小于等于32768字节时，LAM 有优势。少用时百分比平均为6.19%。这最主要是由 LAM 的快速收发造成的，此外与 LAM 结构简明亦有关。包长大于32768字节时，MPICH 有优势，少用时百分比平均为2.16%。这归功于 MPICH 的超长包发送策略。

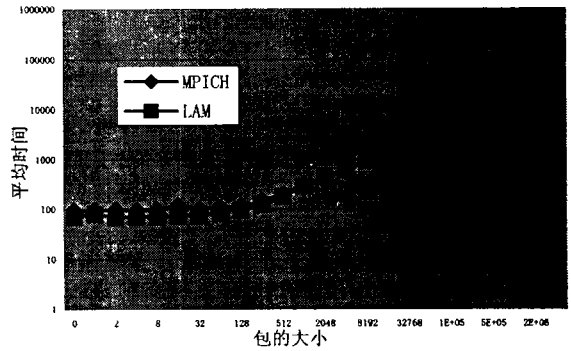

图1　PingPong

SendRecv 模式测试　四进程通信，双方同时发送和接收，其余进程阻塞，结果如图2所示。包长小于512字节时，LAM 绝对优势。少用时超过90%。而且此时 MPICH 开销超过发512字节包时。这主要由 MPICH 短包策略引起。由于小包发送增加了许多拷贝，因此通信的结点越多，这种代价就越明显。包长在512字节和32768字节之间时，二者相差不到20%。包长大于32768字节，LAM 优势逐渐明显（最大超过40%）。

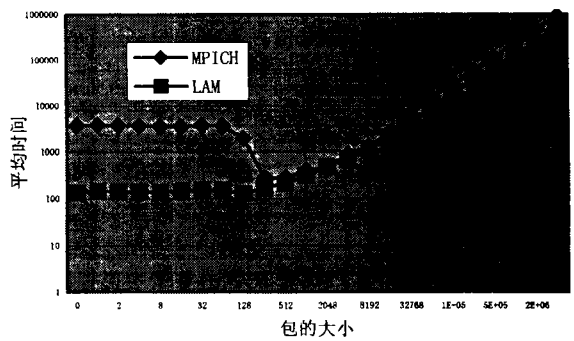

图2　SendRecv

Broadcast 模式测试　所有进程进行 broadcast 通信，结果如图3所示。包长小于等于512字节，LAM 绝对优势，相差在90%以上。当包长在128字节和512字节之间，MPICH 性能突然提升。包长大于512字节，MPICH 有优势。包长4194304，少用时35.0%。原因与上两模式相似。

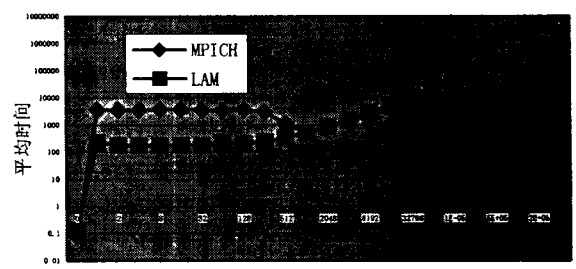

图3　Bcast

3.3 测试小结

评测的结论与以前版本基本一致。应该指出，尽管设备层对性能的影响最大，但在具体接口(如 SendRecv)实现中使用的策略也会对性能有影响。此外，在具体运行中，MPI 程序的差异也可能造成使 MPI 实现表现出有区别的性能。总的来说，如果应用环境是固定的，而且不在多设备等方面有进一步要求，那么可以选择 LAM 使用，因为它的性能更好。如果进行研究与移植，那么 MPICH 是更好的选择。

4. 改进与结论

MPI 库的性能，很大程度上依赖于通信环境。如果要重写设备层来减小软件开销，主要需要考虑1.3.1节的三方面。数据的拷贝更少，调用的层次更少，或通信的协议更先进，都可能导致性能的提高。

MPI 标准规定的非阻塞通信，为体现计算与通信的并行工作。MPICH 和 LAM 中对队列处理函数频繁调用，很好地模拟了计算与通信的并行执行。但单线程的本质，使它们不能够做到更细粒度的并行。大粒度的并行，对通信效率也有影响，比如在 LAM 调用 MPI_Send 时，先建立发送请求对象，然后放入队列，再对队列进行阻塞处理。队列处理函数的阻塞模式，是反复处理队列中的所有请求对象，直到其中没有阻塞标志的对象完成为止。可见，要发送的消息并不一定以最快的速度发出。从计算与通信的角度看，这是由于并行粒度大，导致的通信与通信抢占资源，而不能与计算并列执行。一个直观的解决设计，就是多线程。让队列处理的任务，在一个与用户计算独立的进程中执行，可以提高并行度。MPICH 与 LAM 都给多线程的实现留出了一定的余地，但 MPICH 更加易于实现多线程。这还是在于 ADI 层的抽象设备。MPID_Device 与 API 层的独立，是 MPICH 设计中考虑过的，但没有完成。ADI 层在队列处理函数留有线程锁等代码，这些都为多线程的实现提供了便利条件。多线程设计，最重要的应是解决同步关系。被同步的对象应尽可能地少，因为同步(无论是否经过系统的)会耗费时间。MPICH 给改写人员更大的自由度，可以自己处理请求队列。因为 MPICH 的请求队列完全置于设备层中，所以队列可以交给设备进程。LAM 要做到这一点，需要做更大的改动。此外在进程通信方面，也要考虑像函数调用与返回这样的问题。

MPICH 与 LAM 实现了 MPI1，但没有完全实现 MPI2。现在 MPICH2的 beta 版已经公布，它实现了 MPI2的全部内容。新的 ADI 更巧妙地实现了多设备同时存在并可并行工作，虽然目前的 MPICH2 beta 版还没有实现 TCP 以外部分和多线程，但正式版总会实现多个 ADI 通讯设备的并行处理。现在 MPI 受到广泛欢迎的情况下，相信 MPI 库还会得到不断的进步。

参考文献

1 Message Passing Interface Forum, "MPI: A Message-Passing Interface Standard", 1994. http://www.mpi-forum.org/docs

2 Message Passing Interface Forum, "MPI2: Extensions to the Message-Passing Interface Standard", 1997. http://www.mpi-forum.org/docs/mpi-20.ps

3 Gropp E, et al. A High-Performance Portable Implementation of the Message Passing Interface. Journal of Parallel Computing, 1996,22:789～828

4 Burns G D, Daoud R B, Vaigl J R. LAM: An Open Cluster Environment for MPI. Supercomputing Symposium '94, Toronto, Canada, June 1994

5 Ong H, Farrell P A. Performance Comparison of LAM/MPI, MPICH, and MVICH on a Linux Cluster connected by a Gigabit Ethernet Network. In: the Proc. of the 4th Annual Linux Showcase and Conf. Atlanta, Georgia, USA, Oct. 2000

6 Nevin N. The Performance of LAM 6.0 and MPICH 1.0.12 on a Workstation Cluster: [Technical Report]. Ohio Supercomputer Center, 1996

7 GmbH P. Pallas MPI Benchmarks -PMB, Part MPI-1: [Technical Report]. 2000. http://www.pallas.com

8 Gropp W, Lusk E. An Abstract Device Definition to Support the Implementation of a High-Level Point-to-Point Message-Passing Interface: [Technical Report]. Argonne National Laboratory, USA, 1995

9 Gropp W, Lusk E. MPICH ADI Implementation Reference Manual: [Technical Report]. Argonne National Laboratory, USA, 1995

10 Gropp W, Lusk E. MPICH Working Note: The Second-Generation ADI for the MPICH Implementation of MPI: [Technical Report]. Argonne National Laboratory, USA, 1995

11 Burns G D. The Local Area Multicomputer. In: Proc. of the Fourth Conf. on Hypercube Concurrent Computers and Applications, March 1989

12 Compaq, Intel and Microsoft Corporations. Virtual Interface Architecture Specification. Version 1.0: [Technical Report]. Dec. 1997

13 M-VIA and MVICH, Virtual Interface Architecture Software for Low-Latency, High- Bandwidth, Berkeley Lab VIA Software: [Technical Report]. the Regents of the University of California, 2000

(上接第134页)

种效率和可扩展性较好的算法，引入近似算法，简单而有效解决 IDD 算法中非常重要的候选项目集在各个处理器节点之间的划分问题，尽可能使得各个节点负载平衡。其中，第一种方法为在线算法(online algorithm)，性能比(performance ratio)相对稍差，但是开销非常小，每产生一个候选项目就可以立即处理；第二种方法是离线算法(offline algorithm)，在第一种基础上增加了排序过程，使得性能比大大提高。接下来给出了上述近似算法的性能比的证明。最后，对引入近似算法后的各个算法的复杂度给出了较为详细的分析。

参考文献

1 Stonebraker M, et al. DBMS research at A crossroads: The Vienna update. In: Proc. of the 19th VLDB Conf. Dublin, Ireland, 1993. 688～692

2 Chen M S, Han J, Yu P S. Data mining: An overview from database perspective. IEEE Transactions on knowledge and Data Eng., 1996,8(6):866～833

3 Mannila H, Toivonen H, Verkamo I. Efficient algorithms for discovering association rules. In: AAAI Wkshp. Knowledge Discovery in Databases, 1994

4 Agrawal R, Imielinski T, Swami A. Mining association rules between sets of items in large databases. In: ACM SIGMOD Intl. Conf. Management of Data, 1993

5 Park J, Chen M, Yu P S. An effective hash based algorithm for mining association rules. In: ACM SIGMOD Intl. Conf. Management of Data, 1995a

6 Holsheimer M, Kersten M, Mannila H, Toivonen H. a perspective on databases and data mining. In: 1st Intl. Conf. Knowledge Discovery and datamining, 1995

7 Houstma M, Swami A. Set-oriented mining of association rules in relational databases. In: 11th Intl. Conf. Data Engineering, 1994

8 Agrawal R, Srikant R. Fast algorithms for mining association rules. In: Proc. of the 20th VLDB Conf. Santiago, Chile, 1994. 487～499

9 Savasere A, Omiecinski E, Navathe S. An efficient algorithm for mining association rules in large databases. In: Proc. of the 21st VLDB Conf. Zurich, Switzerland, 1995. 432～443

10 Agrawal R, Shafer J. Parallel mining of association rules. IEEE transactions on knowledge and Data Eng., 1996,8(6):962～969

11 Han E, Karypis G, Kumar V. Scalable parallel algorithms for mining association rules. IEEE Transactions on Knowledge and Data Eng., 1999

12 Ausiello G, et al. Complexity and approximation-Combinatorial Optimization Problems and their Appoximability Properties. Springer-Verlag Birlin Herdelberg, 1999