定时事件触发的精度取决于触发线程检查的周期间隔的长短。假定该周期间隔的长度为 ΔT，意味着每隔 ΔT 时间触发线程将被唤醒，检查是否有到时事件。如果 ΔT 设置得较短，则事件触发的精度比较高，系统实时性比较强，到时事件能及时得到触发，但是如果 ΔT 设置得过短，则需频繁调度触发线程，增加系统负载，反而影响服务的性能。反之，如果 ΔT 设置得过大，则到时事件可能需要经过一段时间的延迟才能等到触发，得不到及时服务，从而影响定时事件服务的质量。由此可见，ΔT 的选择是影响定时事件服务性能的一个关键因素之一。

临界资源的管理也是定时事件服务设计中需考虑的问题。由于多线程的引入，定时队列及定时事件句柄对象均有可能被多个线程共享。但是如果定时队列在插入定时事件句柄对象的同时又实施删除操作，则可能导致队列无法保持有序等现象；如果在取消定时事件句柄对象的周期定时器的同时又触发该到时事件，则有可能导致该对象状态发生混乱。因此定时队列和定时事件句柄对象成为临界资源。一旦一个线程开始对其进行操作，则在该操作结束之前，其它线程就不能对它进行处理。目前实施临界资源互斥的方法有多种，如锁操作，信号量等。

由于多线程的引入和资源的共享与互斥，定时事件服务还可能引起死锁。当若干线程竞争使用资源时，如果每个线程都占有了一定资源，又申请使用已被另一线程占有，且不能抢占的资源，则所有这些线程都将进入封锁状态，不能继续运行下去，从而导致死锁。死锁的产生有多种条件，针对这些产生条件，存在相应的防止死锁的方法，如资源静态分配法，资源顺序使用法等，本文的工作采取资源顺序使用法，即给定时事件服务中的每一个临界资源分配一个唯一的编号，线程申请使用资源时，必须严格按照编号递增的次序进行。这样资源的分配是按序逐步进行，避免了资源分配请求图中循环链的存在，也就防止了死锁的产生。

结束语 基于以上设计思想，我们在 Visigenic 的 ORB 产品 Visibroker 上采用 Java 语言开发并实现了时间服务。在实现中，我们采用层次结构，通过层间时间同步和层内时间同步，取得全局时间同步，并根据层间和层内同步的不同特点分别采用主从式和时间环的概率同步算法，取得较高的同步精度，还基于全局同步时间实现了分布事件的定时服务。

参考文献

[1] OMG, CORBAservices: Common Object Service Specification, OMG Doc. No. 96-10-1, Chapter 14, Time Service Specification

[2] 旷海蓉，CORBA 环境下分布对象服务技术的研究，博士论文，上海交通大学计算机系，1997.12

[3] L. Lamport, Concurrent Reading and Writing of Clocks, ACM Trans. on Computer Systems, Vol. 8 Nov. 1990

[4] N. Vasanthavada, et al., Synchronization of Fault-tolerant Clocks in the Presence of Malicious Failures, IEEE Trans. on Comput., 37(4)1988

[5] Flaviu Cristian, Probabilistic Clock Synchronization, Distributed Computing, No 3, 1989

[6] Alan Olson, et al., Probabilistic Clock Synchronization in Large Distributed Systems, Proc. of the 11th Intl. Conf. on Distributed Computing systems, 1991

[7] Bulent Abali, et al., Clock Synchronization on a Multicomputer, Journal of Parallel and distributed Computing, No. 40, 1997

[8] D. L. Mills, Improved Algorithms for Synchronizing Computer Network Clocks, IEEE/ACM Trans. Networks, 3(3)1995

(上接第107页)

[4] A. Shatdal., et al., Using Shared Virtual Memory for Parallel Join Processing, Proc. of the 1993 ACM SIGMOD, Washington D. C., May, 1993

[5] J. L. Wolf, et al., An Effective Algorithm for Paralleling Hash Joins in the Presence of Data Skew, Proc. of the 1991 ICDE, 1991

[6] M. Kitsuregawa, et al., Parallel GRACE Hash Join on Share-Everything Multiprocessor: Implementation and Performance Evaluation on Symmetry S81, Proc. of the 8th ICDE, Temper, Arizona, Feb. 1992

[7] D. Schneider, et al., A Performance Evaluation of Four Parallel Algorithms in a Shared-nothing Multiprocessor Environment, Proc. of the 1989 ACM SIGMOD, 1989

[8] A. Shatdal, Architectural Considerations for Parallel Query Evaluation Algorithms, the Dissertation for the Degree of Doctor of Philosophy (Computer Sciences), University of Wisconsin-Madison, 1996

计算机科学1998Vol.25№.6

SN和SM体系结构下的各种并行连接算法和并行聚合算法的比较与分析

Comparison and Analysis of Parallel Join Algorithms and Parallel Aggregation Algorithms in Shared-Memory and Shared-Nothing Multiprocessor Environments

方 强 王国仁 于戈 郑怀远

(东北大学计算机系 沈阳 110006)

摘 要 Parallel database system is an advanced issue in database research areas, nowadays. This paper compares the parallel join and parallel aggregation algorithms in details on the two main parallel architectures, shared-memory and shared-nothing multiprocessor. And it also analyzes the main factors which affect the performance of the parallel join algorithms and parallel aggregation algorithms.

关键词 Join, Aggregation, Shared-memory, Shared-nothing, Parallel algorithms

1. 引言

随着数据库应用技术的发展,出现了许多先进的数据库应用系统,如计算机集成制造系统、全球观测信息系统、电子商务信息系统等。它们要求数据库系统既能够处理大量的数据又具有快速的响应能力,因此人们对并行数据库系统进行了许多深入的研究工作。研究工作涉及到许多方面,如物理数据组织策略,并行算法研究,多连接操作的并行查询优化,数据倾斜的处理与动态负载平衡策略等。物理数据组织策略主要研究如何将一个关系均匀地分布到各个处理节点上,主要有四种基本方法:Round-robin 划分策略,Hash 划分策略,Range 划分策略,Hybrid-range 划分策略[1]。多连接并行优化技术主要研究包含多个连接操作的复杂查询的并行优化问题[2]。数据倾斜问题的研究主要是基于连接操作的,具体可以分为四种:初始数据倾斜、选择倾斜、重分布倾斜和连接结果倾斜[3]。动态负载平衡与数据倾斜问题是密切相关的,目前的研究主要集中在如何对并行连接算法进行动态负载平衡,主要的方法有:层次哈希策略[4]、负载共享策略[5]等。并行算法的研究一直是并行数据库系统中的一个主要研究问题,主要集中在并行连接算法和并行聚合算法的研究上,这也是本文要综述的重点。由于本文的综述涉及到两种主要的体系结构,即无共享资源体系结构和共享存储器体系结构,下面对其简单地介绍一下。

在无共享资源(Shared-Nothing,以下简称 SN)体系结构中,没有任何共享的硬件资源(高速通信网络除外),每个节点都有独立的主存储器,CPU,Cache 和磁盘存储器,某一节点的处理机只能直接访问本节点的硬件资源,节点之间通信由高速通信网络实现。这种体系结构的优点是由于处理机间没有共享资源而使得节点间的相互干扰变得最小,这样系统的可伸缩性就很好,能够扩展到上千个节点。其不足之处在于并行算法的实现复杂,比较难于进行动态负载平衡,这样其加速比不是很好。在共享存储器(Shared-Memory,以下简称 SM)体系结构中,多个处理机共享主存储器和磁盘,每个处理机具有自己私有 Cache,多处理机和共享主存之间由高速网络连接。这种体系结构使得并行算法的实现变得比较简单和容易,并且由于所有资源可以共享而使得动态负载平衡问题比较容易解决,这样其加速比较好。但由于多个处理机之间要共享所有的系统资源,因而当处理机的个数增加到一定数量以后,这些处理机对资源的争夺变得非常激烈而使并行算法的性能反而降低,也就是说,SM 体系结构的可伸缩性不是很好。

2. SN 体系结构下的并行哈希连接算法

有很多文章研究了在 SN 体系结构中的并行连接算法,如排序合并连接算法,简单哈希连接算法,GRACE 哈希连接算法,混合哈希连接算法等。我们

在这里主要讨论一下后两种算法。

2.1 并行 GRACE 哈希连接算法[6]

并行 GRACE 哈希连接算法主要分为两个阶段：第一阶段分片，主要是选择一个哈希函数分别对内外关系进行分片操作，并将这些分片写回到磁盘中去。第二阶段连接，要顺序执行内外关系各个分片之间的子连接操作。对于每个子连接来讲，首先要选择一个哈希函数将内关系分片中的所有元组的连接属性值进行哈希，具有相同哈希值的元组被发送到某个节点上，并建立哈希表。当哈希表建立完成后，外关系中对应的分片被读出并用相同的哈希函数来哈希该分片中所有元组的连接属性值，并被发送到相应的节点上去匹配哈希表，并产生连接结果。

2.2 并行混合哈希连接算法[7]

并行混合哈希连接算法与并行 GRACE 哈希连接算法相似，只是在分片阶段，对于内关系来讲，哈希值为0的元组不是写回到磁盘中的第0分片中，而是直接使用第二个哈希函数作用在该元组的连接属性值上，并根据哈希的结果将该元组发送到相应的节点上并建立哈希表。非第0分片中的元组则像 GRACE 算法一样写回磁盘的相应分片中。对于外关系而言，哈希值为0的元组也不写回磁盘，而是直接使用第二个哈希函数作用在该元组的连接属性值上，并发送到相应的节点以匹配哈希表并生成连接结果。当分片阶段结束后，连接阶段与 GRACE 哈希连接算法相类似。并行混合哈希连接算法显然节省了 I/O 开销。因此，从性能上分析，它总是优于并行 GRACE 哈希连接算法。

3. SM 体系结构下的并行哈希连接算法

在 SM 体系结构下的并行连接算法存在一些普遍的问题：(1)由于共享主存，多个处理机的操作必需在共同的区域上进行，这就产生一个问题：对于某个共享存储单元，有多个处理机共享它，这样该单元的存储内容必然能被多个处理机缓存在各自的私有 Cache 中。如果多个处理机以一种循环交错的方式来使用和共享这个存储单元，则系统就必须为维护 Cache 间的一致性付出昂贵的代价。这个问题称之为处理机访问内存单元的局部化问题，即在某个时间段内某个处理机访问某个内存单元的比例越高，则维护 Cache 一致性的代价就越低；(2)Cache 的尺寸较小，往往装不下哈希表的某一桶值的全部元组，这样在元组匹配阶段，会频繁发生 Cache 的缺页中断，使操作时间加长。为克服上述缺点，可将 SN 系统上的某些优点应用到 SM 系统上。下面介绍三种 SM 体系结构下的并行哈希连接算法。

3.1 基于共享存储器的并行哈希连接算法

(1)基于共享存储器的简单并行哈希连接算法

在 SM 体系结构中由于多个处理机共享存储器，这样在共享内存中可以建立一个全局哈希表，哈希表的建立过程是并行执行的：处理机并行扫描每一个内关系的分片，并选择一个哈希函数作用到这些元组的连接属性值上，然后将它们插入到全局哈希表的相应哈希桶中；在哈希表匹配阶段，多个处理机并行地扫描外关系的某个分片，并将相同的哈希函数作用到连接属性值上，然后匹配相应的哈希桶并产生连接结果。由于匹配阶段，多个处理机对共享内存只是读共享，因此各个处理机间的相互干扰较小，但在哈希表建立阶段由于多个处理机要同时修改全局哈希表，因此每个处理机在对哈希表进行修改之前，必须对某一哈希桶进行封锁操作，以便多个处理机能够正确地修改全局哈希表，这就大大地影响了该算法的并行效率。

(2) 基于共享存储器的属性抽取并行哈希连接算法

基于共享存储器的属性抽取并行哈希连接算法与上面的算法基本相似，所不同的是这种算法在哈希表中仅保存内关系中所有元组的连接属性值和指向该元组的元组指针。这样就大大减少了哈希表大小，同时也减少了 Cache 的缺页中断。当连接结果比较小的情况下，这种连接算法要好于基于共享存储器的简单并行哈希连接算法，但是当连接结果大大超过输入关系的尺寸时，在哈希表中的一个元组可能要与外关系中的多个元组产生连接结果，这就要通过哈希表中保存的元组指针来多次访问磁盘，使得 I/O 开销增大，增加了整个并行连接算法的响应时间。

3.2 基于 SN 算法的并行哈希连接

为了克服上述 SM 体系结构的局限，可“借用”SN 体系结构的长处，将其改进用于 SM 的体系结构中。即在主存中为每个处理机开辟一块缓冲区(在共享存储器中开辟一个非共享区)，每个处理机的操作在各自的缓冲区中进行。这是在 SM 体系结构上“模拟”SN 的体系结构的算法。

3.3 基于 Cache 的并行哈希连接算法

上述算法不够好的另一原因是 Cache 的利用率不高，这是由于尽管关系被分片，但每片的尺寸仍很大，以致于每一哈希桶仅有一小部分能放入到 Cache 中，使得 Cache 的命中率很低，因此可将每个

分片进一步分成 Cache 大小的“小片”，使每个“小片”的哈希表都能装入 Cache 中。这实际上是借用了 SN 体系结构的特征，把每个 CPU 的 Cache 视为每个节点的“私有内存”。

通过理论分析和实际系统测试，比较上述三种算法如下：

项目 \ 算法	基于共享存储器的并行哈希连接算法	基于 SN 算法的并行哈希连接	基于 Cache 的并行哈希连接算法
运行时间	对哈希桶的封锁，使其并行效率低；Cache 的命中率低，所耗时间较长	内存中各缓冲区单元通信消耗大；Cache 命中率低，因而所耗时间最长	Cache 命中率高；多次分片有一定的开销
可伸缩性	较 好	较 差	最 好

4. SN 体系结构下的并行聚合算法[8]

在数据库中，聚合操作一般是指对某一关系 R，按元组的分组属性(Group By)所进行的求和，求最大最小值或求平均值等操作。可见，操作结果的元组数依赖于 Group By 子句的选择率，即分组属性值的个数与输入关系 R 的元组数的比率。在 SN 体系结构上，聚合算法的关键问题是处理好内存消耗和通信开销这两方面。在 SN 体系结构中并行聚合算法可分为以下几种：

4.1 集中式二阶段并行聚合算法

对于一个关系而言在物理数据库设计阶段被分为若干个分片，每个分片存储于不同的节点上。集中式二阶段并行聚合算法可分为两个阶段来执行：第一个阶段称为局部聚合，它是在各个节点上进行一个集中式聚合操作。在第二个阶段，各个局部聚合结果都发送到一个中央节点(称为协调者)进行最后的全局合并，生成最终的全局聚合结果。

该算法的优点在于当 Group By 子句的选择率较小时，由于各个局部聚合结果的聚合组数较少，这样就减少了算法的通信开销。但是随着 Group By 子句选择率的不断增加，通信开销也将随之增加，中央协调者的工作负载也就越来越重，并最终可能成为并行算法的性能瓶颈。

4.2 层次合并并行聚合算法

层次并行聚合算法对集中式二阶段并行聚合算法进行了改进，也分两个阶段执行：第一阶段为局部聚合阶段，与集中式二阶段并行聚合算法相似；第二阶段称为层次合并阶段。与集中式二阶段并行聚合算法不同，不是将各个节点的聚合结果发送到一个中央协调者，而是分层次并行地进行部分聚合结果的合并，并得到中间合并结果，这些中间结果可能被进一步并行地合并为新的中间结果或者合并为一个全局聚合结果。显然该算法在性能上作了改进，减轻了合并节点的工作负担，但它并不能最终解决性能瓶颈问题，因为当 Group By 子句的选择率足够大时，层次合并阶段亦会成为该算法的性能瓶颈，只是该算法性能瓶颈的出现比集中式二阶段并行聚合算法来得晚些。

4.3 两阶段并行聚合算法

两阶段并行聚合算法也分为两个阶段：第一阶段为局部聚合阶段，在这个阶段各个局部节点在各自的节点上进行集中式的聚合操作，并将局部聚合结果通过选择一个哈希函数作用到 Group By 属性上，将局部聚合结果散列到各个节点上。第二个阶段为全局聚合阶段，这个阶段主要是合并被散列到该节点上的局部聚合结果。由于对局部聚合结果进行了重新散列，所以全局聚合阶段的聚合结果不需要进行合并，这样全局聚合阶段是在各个节点上并行进行的，避免了前两种算法的性能瓶颈问题。

4.4 再分布并行聚合算法

基本思想是：先将聚合操作的子操作的元组通过选择一个哈希函数并作用到 Group By 属性值上，将这些元组再分布到不同的节点上，在每个节点上有一个聚合操作，该操作接收再分布操作符发送过来的元组并进行聚合操作。由于对来自于聚合操作组的输入元组直接进行了散列，所以各个节点的聚合操作结果不需要进行合并。

该算法与两阶段并行聚合算法相比较有以下几个特点：(a)再分布并行聚合算法散列的是各个局部分片或聚合操作的输入，而两阶段并行聚合算法散列的是各个局部聚合结果，也就是说前者在网络中传输的是子分片中的元组，而后者传输的是局部聚合的结果元组，因而后者的网络通信开销较少；(b)两阶段并行聚合算法有两级聚合操作，即局部聚合操作和全局聚合操作，而再分布并行聚合算法只有

一级聚合操作；(c)当 Group By 子句的选择率很低时，使得再分布并行算法在初始散列之后，由于分组数少可能造成有些处理机没分配到任务而处于空闲，从而使处理机的利用率较低；(d)当 Group By 子句的选择率高时，两阶段并行算法性能不如再分布算法。因为两阶段并行聚合算法在进行局部聚合时，对于同一分组属性在不同的节点内存中都可能有其局部聚合结果的入口；而再分布并行聚合算法进行散列后，每个分组属性在系统中只存在一个聚合结果。所以两阶段并行聚合算法对内存总的占用量较大，易使某些节点发生内存不足而需要进行额外的 I/O 操作，算法性能也随之迅速下降。而再分布并行聚合算法不需要局部聚合操作，也不会由于内存不足而进行 I/O 操作，因此其性能将越来越好。因此当选择率超过某一值后，再分布并行聚合算法的性能要优于两阶段并行聚合算法。

4.5 改进算法

从上述分析可知，每种算法都不可能在 Group By 子句选择率的整个范围内都有较好的性能，但非常幸运的是，两阶段并行聚合算法和再分布并行聚合算法在 Group By 子句选择率的整个范围内是互补的，即在选择率较小时，两阶段并行聚合算法的性能比较好，而当选择率超过某一范围后再分布并行聚合算法的性能相对较好。下面三种改进算法是针对这一问题而提出的。

(a)基于抽样的并行聚合算法　随机的抽取某几物理页上的关系元组进行抽样测试，估算 Group by 子句的选择率范围，从而决定使用两阶段并行聚合算法，还是用再分布并行聚合算法。

(b)自适应的两阶段并行聚合算法　基本思想是：所有节点开始使用两阶段并行聚合算法，当某个节点检测到它的哈希表已占满内存并可能发生溢出时，它就用哈希函数将部分累积结果散列到相应节点，进行全局聚合，从而释放部分内存，随后，读入未处理的元组用同样的哈希函数处理，这就由最初的两阶段并行算法切换到再分布并行算法。这种算法的实现比较复杂，运行一段时间后，每个节点根据本场地的初始关系的情况，采取不同的运行机制，因此在某一时间点，有可能部分节点使用两阶段并行聚合算法，而另一部分使用再分布并行聚合算法。

(c)自适应的再分布并行聚合算法　开始使用再分布并行聚合算法，在执行过程中，某节点检测到它所处理的分组数很小，CPU 的利用率较低，它就向其它所有节点发出一条"阶段结束"的消息，其它节点收到该消息后，同时切换到自适应两阶段并行聚合算法。

总结上述五大类算法，比较列表如下：

算法 / 项目	集中式二阶段并行聚合算法	层次合并并行聚合算法	两阶段并行聚合算法	再分布并行聚合算法	改进算法
Group By 子句选择率较小	通信开销小，性能较好	通信开销较小，并行性较好	网络传输代价小，性能好于集中式二阶段算法和层次合并算法	处理机利用率低，性能较差	性能好
Group By 子句选择率较大	通信开销增大，协调者成为算法的性能瓶颈	仍可能出现性能瓶颈	网络传输代价大，内存占用量大，I/O 操作次数增多，性能降低，不存在性能瓶颈	网络传输量小，内存占用量小，性能较好	性能好

5. SM 体系结构下的并行聚合算法

在 SM 体系下执行聚合运算与 SN 结构的情况相类似，常用的传统算法有：简单并行聚合算法和两步并行聚合算法。前者在共享内存中对关系 R 建立全局哈希表，并采用上锁机制。后者是在 SM 体系结构下实现 SN 体系结构下的两阶段并行聚合算法，即分为局部聚合和全局聚合，在此过程中，仍需使用上锁机制，但其冲突要少得多。

分析上述算法，在关系的 Group By 子句选择率较低时，简单并行聚合算法性能较差，因为分组少，则哈希桶的数目较少，所以多个处理机并行工作时，很容易为争用这几个哈希桶而发生冲突；当 Group By 子句的选择率较高时，多个处理机试图访问同一哈希桶的可能性会大大减小。两步并行聚合算法很好地克服了简单并行聚合算法的缺点，但在 Group By 选择率很大时，第二步开销明显增大。所以，我们必须权衡简单并行聚合算法的上锁冲突开销和两步并行聚合算法的额外聚合工作的开销，根据具体情况选择合适的方法。即在 Group By 子句选择率小时

采用两步并行聚合算法，而选择率大时采用简单并行聚合算法，其切换点由具体系统来决定，包括考虑上锁和等待解锁的时间，以及聚合操作的耗费等。自适应的两步并行聚合算法和自适应的简单并行聚合算法就是基于这种思想提出的。在实际性能分析中，这两种适应性算法避免了传统算法的不足，因此有很好的性能。

结论 综上所述，SN 和 SM 体系结构下的并行算法并不是完全独立的。对于并行连接算法，在 SN 体系结构下，本文主要介绍了传统的并行 GRACE 哈希连接算法和并行混合哈希连接算法。由于 SN 系统的各节点间处理干扰小和较好的规模可扩展性，因此在 SM 连接算法中可以借用一些 SN 算法的处理方法。并行聚合算法则主要权衡并行处理的通信开销和各个节点对内存的需求。在 SN 和 SM 结构下，各种传统算法的性能优劣在整个 Group By 子句的选择范围内形成互补，因而将其有机的结合起来，形成自适应算法，其性能大大好于传统算法。

参考文献

[1] D. DeWitt, et al., Parallel Database Systems: The Future High Performance Database Systems, Communication of ACM, 75(1)1986

[2] Y. E. Ioannidis, et al., Left-Deep US Bushy Trees: An Analysis of Strategy Space and Its Implications for Query Optimization, Proc. of the 1991 ACM-SIGMOD, Colorado, May, 1991

[3] C. B. Walton, et al., A Taxonomy and Performance Model of Data Skew Effects in Parallel Joins, Proc. of the 17th VLDB, Barcelona, Sept. 1991

（下转第102页）

新书介绍

小波分析与信号处理
——理论、应用及软件实现

本著作全面系统介绍了小波分析的基本理论和最新研究成果，重点介绍小波分析的应用研究成果，并通过软件实现来检验应用效果。全书分为三篇：第一篇是小波理论，包含8章的内容，如小波分析的发展历史及文献综述、准备知识、多分辨分析与共轭滤波器、连续小波变换、最佳小波基的构造及算法、二维母小波的构造、框架与样条小波理论、时间频率分析；第二篇是小波应用，包含12章的内容，详细介绍小波分析在图像压缩、流体力学、工业CT、故障诊断、语音分割、数学物理、地球物理勘探、医学细胞识别、线性系统、神经网络等多方面的应用；第三篇是小波软件，将第一篇和第二篇的重要算法用 C 语言实现，且全部在微机上调试通过，读者可直接应用或稍加修改即见成效。本书已列入"全国高技术重点图书"出版计划。

著名科学家、中国科学院院士林为干教授为本书作序，著名计算机专家、重庆大学计算机研究所所长、博士生导师陈廷槐教授担任本书的学术顾问。

本书的鲜明特点是强调应用、突出重点、由浅入深、算法编程。本书的读者对象主要是从事信号分析、信号获取与处理、图像处理、通讯理论、数学、物理、计算机、医学、化学、石油地质勘探、机械工程等方面的研究人员、大学教师、研究生、大学生，尤其适合专门从事处理突发性问题的工程技术人员，对于有兴趣学习新学科、高科技知识的人来说，本书也是很好的入门图书。

全书共60余万字，由重庆出版社科学学术著作出版基金资助以精装形式出版。本书定价48.00元/本，外埠邮购另加15%邮费，共计55.2元/本，欲购书者请从邮局汇款至：(400042)重庆后勤工程学院数学系 李建平收。并请在汇款附言上注明购书的册数。汇款到后，书立即用挂号寄出。李建平联系电话：(023)65103199(办)，(023)68756948(宅)，BP：129(8)传 8180518。