THSORT:单机并行排序算法*

施 遥, 张 力, 刘 鹏+

(清华大学 计算机科学与技术系,北京 100084)

THSORT: A Single-Processor Parallel Sorting Algorithm

SHI Yao, ZHANG Li, LIU Peng+

(Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)

+ Corresponding author: Phn: 86-10-62782530, Fax: 86-10-62771138, E-mail: pengliu@ieee.org

http://hpclab.cs.tsinghua.edu.cn/~liupeng/

Received 2002-11-19; Accepted 2002-12-10

Shi Y, Zhang L, Liu P. THSORT: A single-processor parallel sorting algorithm. *Journal of Software*, 2003,14(2):159~165.

Abstract: Sorting is an important operation of transaction processing. It is a relatively mature field, as many algorithms for memory sorting, disk sorting and parallel sorting have come forth in the past decades. In this paper, the sorting algorithm is studied from a thoroughly different standpoint, and the THSORT (Tsinghua SORT), a parallel sorting algorithm on a single computer, is brought forward. THSORT uses several processes to control different components of a computer, which enables the data input, sorting and output to be run concurrently, and thus greatly enhances the parallelism and efficiency of the hardware. Experimental results based on a computer with two RAIDs (redundant array of inexpensive disks) indicate that THSORT has almost doubled the performance of NTSORT (new technology SORT), a famous sorting program. Moreover, THSORT has won the 2002 PennySort competition and is still holding the world record in the Daytona category.

Key words: parallel I/O; single-processor parallel sorting; THSORT (Tsinghua SORT); PennySort

摘 要: 排序是计算机事务处理的重要操作之一.前人已经就内部排序、外部排序和并行排序提出各种方法.从一种全新的视角研究了排序算法,提出一种在单机上实现的并行排序算法 THSORT(Tsinghua SORT).它用多个进程分别控制不同的硬件部件,使输入、排序和输出能够同时进行,从而大大提高了硬件部件的并行性和运行效率.在带有双磁盘阵列的硬件平台上进行的测试表明,THSORT 的性能达到了 NTSORT(new technology SORT)的 1 倍左右,并成为 2002 年 PennySort(Daytona 类)世界排序纪录的保持者.

关键词: 并行 I/O;单机并行排序;THSORT(Tsinghua SORT);PennySort

中图法分类号: TP301 **文献标识码**: A

排序是计算机处理的基本功能,应用范围极为广泛,特别是在数据库系统、数据分析和数据挖掘中被大量

* Supported by the Doctor's Scientific Research and Innovation Foundation of Tsinghua University of China(清华大学博士生科研创新基金)

第一作者简介: 施遥(1982－),男,江苏海门人,主要研究领域为高速磁盘阵列,高速网络服务器.

使用.

对小数据量(MB 量级)进行排序,可以采用已经研究得比较透彻的内部排序算法,如 Quick Sort[1],Flash Sort[2],Proportion Split/Extend Sort[3],分段快速排序[4]、基数分配链接排序[5]等.内部排序基于内存空间条件不受限制的理想化情况,并不适用于大规模数据排序.

在对大规模数据(GB 量级)进行排序时,必须使用外部排序算法[6](包括多磁盘外部排序[7]).它们首先进行分块内部排序,然后将有序的块进行归并.在进行分块内部排序时,每一个数据块都要经过“读入数据-排序-写出数据”的处理流程.由于在读入和写出数据阶段 I/O 设备运转而 CPU 大多在闲置,排序阶段 CPU 的负荷极重而 I/O 设备在闲置,分块内部排序的设备利用率低,排序效率不高.也就是说,外部排序的算法会与 I/O 吞吐产生一定的依赖关系,不再是单纯的数学算法问题,而是算法与硬件系统相结合的问题.所以,如何在高效的算法支持下,协调计算机各部件的使用,以提高整个排序过程的速度,是亟待解决的问题.

本文提出了可以在相对廉价的单台计算机上实现的并行外部排序算法,充分利用了单机上不同硬件部件的并行性,运行多个不同的进程,在某一时刻,每个进程处理不同的事件,以实现排序算法的并行处理,使得整个排序过程对于计算机各部件的利用率得到相当大的提高,最终使排序速度得到显著提升.这样,就可以用较低的成本,实现大规模数据的高效排序.基于上述思想,我们设计了单机并行排序算法,并以此实现了 THSORT (Tsinghua SORT)排序软件.实验结果表明,在双磁盘阵列计算机上,它的排序速度可以达到串行排序程序的 1 倍左右.THSORT 参加了 2002 年度 PennySort[8,9]世界排序比赛,并成为 Daytona 类别的世界记录保持者[10].

对海量数据(TB 量级)进行排序,往往需要由拥有多个处理器和多个独立内存空间的并行计算机来完成[11].并行排序既可以基于机群 (如 Berkeley 的 NOW-Sort) [12],也可以基于多处理器系统[13],这两者本质相似,但造价都比较昂贵.本文所述的算法完全可以扩展到并行计算机的环境中,用以提高每一个处理机的排序效率,从而提高整体排序效率.

1 算法分析与设计

THSORT 分为两个步骤:

(1) 形成局部有序的数据块:将数据顺序分成不超过内存大小的小块读入,对每一个小块进行排序,并将结果分别保存到临时文件.

(2) 归并数据块:将第 1 步生成的所有临时文件进行归并排序,得到全局有序的数据.

这两个步骤在算法设计上可以说是独立的,但是对于算法优化来说,两者之间是相互制约的.下面我们分别介绍这两个步骤的算法设计,然后再对它们进行综合分析.

1.1 分块排序

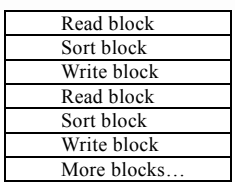

Fig.1 Single process sorting

图 1 单进程的情况

设总数据量为 N,分块后每次处理的数据量为 B(不妨设 $N=nB$),则进行一次分块排序所需要的时间为 $t_0(B)$,由读入数据块(read block)、排序块(sort block)、保存数据块(write block)这 3 个步骤所需的时间决定.这三者均是 B 的函数,分别记作 $t_r(B),t_s(B),t_w(B)$.如果只用单个进程来进行处理,即将以上的 3 个步骤串行执行(如图 1 所示),那么显然有

$$t_0(B)= t_r(B)+ t_s(B) +t_w(B),$$

而完成全部数据块的排序需要的总时间将为

$$T_0(B)=nt_0(B)=n[t_r(B)+ t_s(B)+t_w(B)].$$

然而,由于 Read Block,Sort Block,Write Block 三者占用的系统资源各不相同,如果我们把排序前的数据和排序后的数据放置在两个不同的硬盘上,Read Block 和 Write Block 就可以同时进行,并且不争夺 I/O 资源.同时,这两个模块占用 CPU 资源很少,可以和 Sort Block 并行运行.因此,我们可以通过采用单机并行排序算法来提高效率.

由于上述 3 种模块(block)自身是不相容的,即在同一时间内同时运行的两个 Read Block(或者 Sort

Block,Write Block)将争夺系统资源,因此合理的并行算法必须遵守以下的规则:

第 K 个数据块的 Read Block 在第 K–1 个数据块的 Read Block 结束后开始;

第 K 个数据块的 Sort Block 在第 K–1 个数据块的 Sort Block 结束后开始;

第 K 个数据块的 Write Block 在第 K–1 个数据块的 Write Block 结束后开始;

第 K 个数据块的 Sort Block 在第 K 个数据块的 Read Block 结束后开始;

第 K 个数据块的 Write Block 在第 K 个数据块的 Sort Block 结束后开始.

引入最后两条是要求对一个数据块而言,3 个 Block 必须满足读、排序、写的逻辑顺序.

根据以上的 5 条规则,对于给定的 B,我们可以建立以 $3n$ 个模块(Read,Sort 和 Write Block 各 n 个)执行作为顶点的 PT 图(potential task graph)$G(V,E)$:

令顶点 X_K,Y_K,Z_K 分别表示第 $K(K=1,2,\ldots,n)$个数据块的 Read Block,Sort Block,Write Block 的开始,$V=\{X_K\}\cup\{Y_K\}\cup\{Z_K\}$.令 $E=\{(X_K,Y_K),(Y_K, Z_K),(X_K, X_{K+1}),(Y_K,Y_{K+1}),(Z_K,Z_{K+1})\mid K=1,\ldots,n\}$,边权为

$$w(X_K,Y_K)= w(X_K,X_{K+1})=t_r(B),$$
$$w(Y_K,Z_K)= w(Y_K,Y_{K+1})=t_s(B),$$
$$w(Z_K,Z_{K+1})=t_w(B),$$

如图 2 所示.

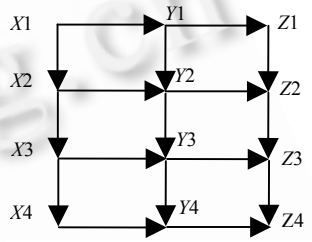

Fig.2 PT graph for n=3

图 2 n=3 的 PT 图

由关键路径算法,就能够得到最优的并行执行方案和以 B 为函数的总耗时 $T(B)$.再对 B 的不同取值作最优化,可求出使得 $T(B)$取得最小值的最优数据块大小 B'.然而,我们在实际算法实现时,采用的是另一种近似最优的并行算法.考虑当 B 的取值满足条件:

$$t_r(B)\approx t_s(B)\approx t_s(B)=t'.$$

我们简单地采用 3 个进程并行运行的方法来对源数据进行排序(如图 3 所示),使得 CPU 和 I/O 资源得到充分地利用.在 3 个进程都开始运行之后,任何一个时刻都恰有一个 Read Block、一个 Write Block 和一个 Sort Block 在并行运行.由于没有任何等待时间,这恰好是当 $B=B'$时的由关键路径算法求得的最优并行执行方案.在这种情况下,3 个进程并行算法的总耗时可以简单地表示成

$$T'= t_r(B')+nt_s(B')+t_w(B')=(n+2)t'.$$

Thread A	Thread B	Thread C
Read block		
Sort block	Read block	
Write block	Sort block	Read block
Read block	Write block	Sort block
Sort block	Read block	Write block
Write block	Sort block	Read block
	Write block	Sort block
		Write bock

Fig.3 Parallel sorting of three processes

图 3 三进程并行排序

再考察并行排序算法总耗时 T 的下限.由于 I/O 读写的速度比内存里元素的交换慢很多,因此一般说来有

$$t_w(B)\geq t_r(B)>t_s(B).$$

同时,由于所有 Write Block 的消耗总时间是

$$nt_w(B)\approx t_w(nB)>t_w(N),$$

而两个 Write Block 之间是不能并行的,因此并行算法的总耗时满足

$$T\geq t_w(N),$$

而 $T'=(n+2)t'=t_w(N)+2t'$与 T的下限很接近(仅仅是 T_0的 1/3 左右),因此选择三进程的并行算法不失为一种简单、有效的方法.

但在 $T_s(B'),T_r(B'),T_w(B')$不尽相等的情况下,采用三进程的并行方法将出现等待的情况.一般说来,Sort Block

采用的不是线性算法,而 I/O 的时间开销是与数据量呈线性关系的.所以,当 B 较小时,$T_s(B)$耗时比 $T_r(B)$和 $T_w(B)$少;在 B 增大时,$T_s(B)$耗时将增加以至于超过 $T_r(B)$和 $T_w(B)$.

在每个数据块的 Sort Block,我们采用快速排序算法,在源数据随机排列的情况下,这一算法的平均时间复杂度为 $O(n\log_2 n)$,是基于交换的排序事件复杂度的下限.而在 Read Block 和 Write Block 中,我们尽量优化代码,选择高级语言(GNU C)能够提供的最快速的磁盘操作.

1.2 多路归并

多路归并算法需要将所有临时文件中有序的数据合并到最终的输出文件中并保持有序.这个阶段,本文采用了比较高效的堆排序算法[14].

同样,按照充分利用系统资源的原则,归并算法分为两个并行的进程:读进程与写进程.读写进程共享同一块内存,读进程将临时文件内的数据补充到内存中,而写进程则从内存中取出最小元素归并到输出文件中(如图 4 所示).

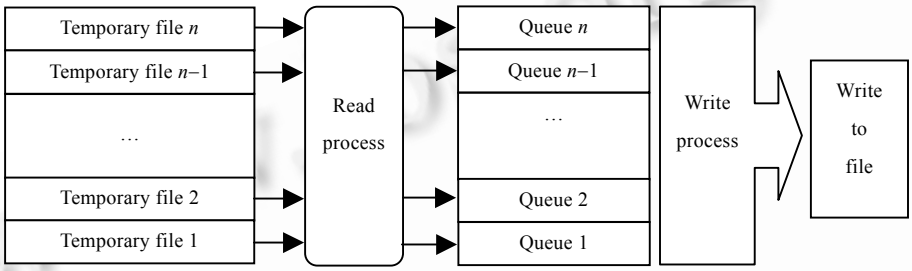

Fig.4 Merging process

图 4 归并流程

设共有 n 个临时文件需要进行归并,则将内存划分为 n 个循环队列,作为对应临时文件在内存中的缓存,每个队列大小为 M;在初始化时,将第 k 个临时文件内的数据全部或者部分(当文件长度大于 M 时)装入到队列中,同时,

(1) 判断某文件是否被读完,若是,置该文件读完标记 $flag_i$=1,否则 $flag_i$=0.

(2) 设置每个内存块中的数据量 $size_i$ 等于读入的数据量.

初始化结束后,就由读进程负责将临时文件中的数据不断向内存块中进行补充,直到所有临时文件被读完:

(1) 取第 k 个数据块,使得 k 满足:

$$size_k = \min_{flag_i=0}\{size_i\}.$$

(2) 读取临时文件 k 加入队列 k 中,直至文件读完或队列 k 满.若文件 k 已读完,置 $flag_i$=1.

(3) 若所有文件读完,退出;否则转(1).

与此同时,写进程将 n 个内存块中的数据归并到输出文件中.由于临时文件的数量可能很多,如果通过逐个从 n 个内存块中取出最小元进行归并效率是很低的.为了提高效率,我们采用了二叉堆数据结构来求最小元,其插入结点和调整的复杂度均为 $O(\log_2 n)$,而求最小元的复杂度为 $O(1)$.

写进程的算法可以描述如下:

(1) 以每个队列头的数据,组成一个最小二叉堆,即每个父结点的关键字均小于子结点的值.

(2) 输出最小元(堆的根结点),将该数据从二叉堆和对应队列中删去.

(3) 由于需要与读进程同步,此时需要考虑以下 3 种情况(设最小元属于队列 k):

(a) 若队列 k 不为空:将队列首的数据入堆作为根节点,调整堆;

(b) 若队列 k 为空且文件 k 未读完:必须等待临时文件中的数据读入,再将队首数据入堆作为根节点,调整堆;

(c) 若队列 k 为空且文件 k 已经读完:若此时堆为空,则结束写进程;否则,取堆的任意一个叶结点作为新

的根节点,调整堆.

(4) 转(2).

堆的递归调整方法如下:

(1) 令 i 为根结点;

(a) 若结点 i 无子结点,结束堆调整;

(b) 若结点 i 有子结点,且其关键字均大于结点 i,结束堆调整;

(2) 选择关键字值较小的子节点(设为结点 j),与结点 i 交换,令 $i=j$,转(1).

由于读写进程是并行的,因此归并的总耗时等于两进程耗时的较大值.读进程的总耗时大约等于 $nt_r(B)$(实际上由于队列控制的开销会大于此值);写进程的总耗时由写入文件的时间和二叉堆操作的时间组成,由以上算法可知,其大约等于 $Nt_h(n)+nt_w(B)$(其中 $t_h(n)$是一次堆调整的耗时,它是堆的规模即队列数 n 的函数).

因此,归并算法的总耗时为

$$T=\max\{n\cdot t_r(B),N\cdot t_h(n)+n\cdot t_w(B)\}$$
$$=\max\{t_r(N),N\cdot t_h(n)+t_w(N)\},$$

即 T 是 n 的函数.一般有 $t_w(N)\geq t_r(N)$,因此归并算法总耗时一般是随 $t_h(n)$的增大(即 n 的增大)而增加的.

1.3 算法综合分析

综合以上两个步骤的分析,在分块排序中,B 取值越小(数据块大小越小),分块排序的耗时就越少;在多路归并中,n 越小(数据块的数目越少),归并算法的耗时就越少.

比较两部分算法可以发现,在总数据量 N 一定的情况下,分块大小 B 对于整个排序过程中两个阶段效率的影响是矛盾的:块分得大,B 大,n 小,分块排序慢而多路归并快;块分得小,B 小,n 大,则分块排序快而多路归并慢.在这两者之间寻找一个最优的平衡点就成为我们进行性能优化的一个主要任务.

通过测试,在我们配置的机器上,处理 10G 左右数据,每一块的大小约在 20M~50M 之间,能够达到最高的效率.

2 具体实现

2.1 硬件配置

THSORT 测试平台的硬件配置见表 1.

Table 1 Hardware configuration of THSORT testing platform

表 1 THSORT 测试平台的硬件配置

Mainboard	Abit KR7A-RAID(with HPT 372)
CPU	AMD ATHLON XP1700+
Hard Disk	IBM 40GB HDD ATA100 IDE 7200RPM×4
Memory	KINGMAX PC2100 256MB DDR SDRAM×2

我们将 4 块硬盘两两组合配置为 A,B 两组磁盘阵列,其中,A 组磁盘阵列存储源数据及最终排序所得数据;B 组磁盘阵列存储算法第 1 阶段所得出的分块有序数据.这样使用读操作与写操作能够得以同时进行.在算法第 1 阶段分块排序时,从 A 组磁盘阵列读出数据、将排序完成的数据写入 B 组磁盘阵列;在算法第 2 阶段归并排序时,从 B 组磁盘阵列读出有序数据块、进行归并操作后写入 A 组磁盘阵列中.

2.2 程序实现说明

我们测试了 Windows 与 Linux 的 I/O 速度,发现在 I/O 速度上(特别是写操作),Linux 具有较大的优势,因此 THSORT 是基于 Linux 实现的.

在 Linux 环境下,使用系统 VIPC 的方法,开辟一块虚拟内存(在具体实现中,为 500M 字节),供多个进程公共使用.由于 Linux 的 Write Buffer 需要占用一定的内存,所以不宜将所有内存都辟作共享内存.

在程序的编写过程中,根据机器的配置,需调整分块排序算法中每一块的大小,使得每一数据块的排序与

读、写操作时间基本相等.

2.3 测试结果比较

我们将 THSORT 与著名的排序软件 NTSORT(new technology SORT)在同一台机器上作了性能比较.通过测试发现,THSORT 较 NTSORT 约快 1 倍左右,如图 5 所示.这说明在特定的硬件配置下,单机并行算法有相当大的开发价值与潜能.

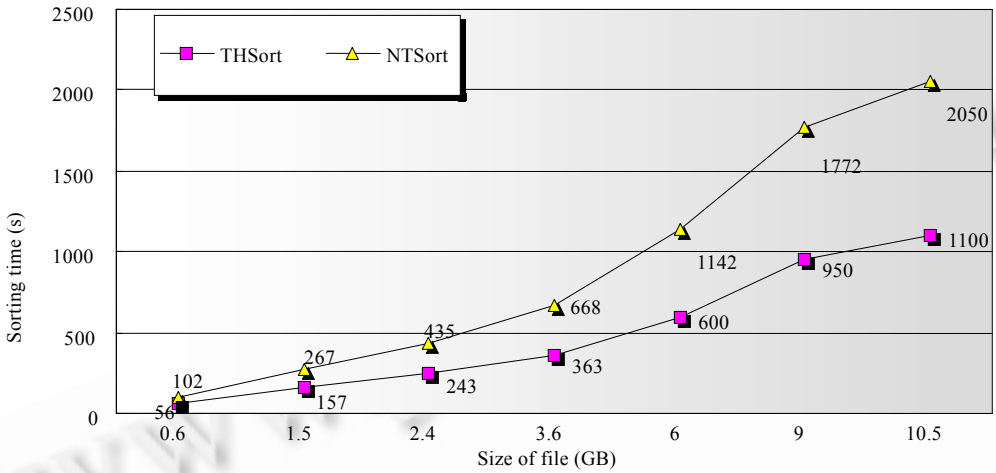

Fig.5 THSORT vs. NTSORT in performance

图 5 THSORT 与 NTSORT 的性能比较

从算法分析中可以看到,在分块排序时,THSORT 的速度约为普通串行排序方法的 3 倍(即 THSORT 处理时间约为普通串行排序方法的 1/3);而归并排序时,THSORT 的速度约为普通串行排序方法的两倍(即 THSORT 处理时间约为普通串行排序方法的 1/2).因此,从理论上讲,对于相同的数据量,THSORT 的速度应是普通串行排序方法的 2~3 倍.

但从测试结果来看,THSORT 的速度比理论值略慢一些.究其原因,主要是由于受到硬件条件的限制:

(1) CPU 的限制.在算法分析中,我们将读写操作所需占有的 CPU 资源忽略不计.而事实上,读写操作需要占用一定的 CPU 资源(一般为 10%以下).在程序运行过程中,我们观察到,CPU 基本上是满负荷运作,这说明 CPU 在一定程度上成为了程序的瓶颈,但又从另一个方面说明了程序对硬件的利用是比较充分的.

(2) 两组磁盘阵列同步读写只是理想化的情况.实际上,两组磁盘阵列的读写操作尽管较之串行操作来得快,但很难达到真正完全的同步,还是会有一定差距的.这是由硬件技术的局限所造成的,但随着硬件技术的发展,同步性能将会不断提高,这样程序便更能与算法在理论上相吻合了.

尽管由于上述原因使程序效率有了一定程度的降低,但其速度仍能达到普通串行排序程序的两倍,这说明本算法在理论上和实践中都是比较有价值的.

3 结 语

THSORT 较充分地利用了计算机的主要硬件部件,从而大大缩短了串行排序所需的时间.多进程的交替并行 I/O,使得读、写与排序得以同步进行,有效地利用了原来闲置的资源,大大提高了算法的效率.在算法的分块排序中使用快速排序,而在多路归并排序中使用堆排序.这两部分在实现与设计上是独立的,但在时间效率上是相互依赖与矛盾的.最后我们给出了本算法的实现,并进行测试,得到了很好的结果,达到了期望要求.在算法的实现中,必须紧密结合计算机本身的情况进行算法的优化和运行参数的选取,以充分发挥硬件部件的潜能.

References:

[1] Knuth DE. The Art of Computer Programming (Volume 3/Sorting and Searcing). 2nd ed., New York: Addison-Wesley Publishing Company, 1973. 114~116.

[2] Neubert KD. The flashsort algorithm. Dr. Dobb's Journal, 1998,23(2):123~129.

[3] Chen JC. Proportion extend sort. SIAM Journal on Computing, 2001,31(1):323~330.

[4] Tang XY. Fast sorting method of separating segment. Journal of Software, 1993,4(2):53~57 (in Chinese with English Abstract).

[5] Wang XY. A new sorting method by nase distribution and linking, Chinese Journal of Computers, 2000,23(7):774~777 (in Chinese with English Abstract).

[6] Elmasri R, Navathe SB. Fundamental of Database Systems. 2nd ed., Redwood City, CA: Benjamin/Cummings Publishing Company, 1994. 83~84.

[7] Feldman MB. Data Structures with Ada. Reston: Reston Publishing Company, 1985. 290~300.

[8] Helmkamp B, McCready K. 2000 Performance/Price sort and PennySort. Stenograph LLC Company, 2000. http://research.microsoft.com/barc/SortBenchmark/Y2000_PennySort.doc.

[9] Gray J, Coates J, Nyberg C. Performance/Price sort and PennySort. Technical Report, MS-TR-98-45, Microsoft Research, 1998.

[10] Sort Benchmark. http://research.microsoft.com/barc/SortBenchmark/.

[11] Shi HM, Schaeffer J. Parallel sorting by regular sampling. Journal of Parallel and Distributed Computing, 1992,14(4):361~372.

[12] Arpaci-Dussea AC, Arpaci-Dusseau, RH, Culler DE., Hellerstein JM., Patterson DA. High-Performance sorting on networks of workstations. In: Peckham J. ed. Proceedings of the 1997 ACM SIGMOD Conference. Tucson: ACM Press, 1997. 243~254.

[13] Taniar D, Rahayu JW. Sorting in parallel database systems. In: Proceedings of the 4th International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region. Beijing: IEEE Computer Society, 2000. 830~835.

[14] Wegner LM, Teuhola JI. The external heapsort. IEEE Transactions on Software Engineering, 1989,15(7):917~925.

附中文参考文献:

[4] 唐向阳.分段快速排序法.软件学报,1993,4(2):53~57.

[5] 王向阳.任意分布数据的基数分配链接排序算法.计算机学报,2000,23(7):774~777.

****