无存储器冲突的并行快速排序算法*

管 丽

(北京师范大学 北京 100875)

摘要 本文在一个 EREW PRAM (exclusive read exclusive write paralled random access machine)上提出一个并行快速排序算法,这个算法用 k 个处理器可将 n 个项目在平均 $O((n/k+\log n)\log n)$ 时间内排序. 所以平均来说算法的时间和处理器数量的乘积对任何 $k\leqslant n/\log n$ 是 $O(n\log n)$.

关键词 并行算法,排序算法,快速排序算法.

对于并行排序算法的研究一直是并行计算机领域的一个活跃的分支.[1,2]并行随机存储器 PRAM(parallel random acess machine)则是常常用来作为研究并行计算的模式. PRAM 由一些处理器组成,每个处理器有自己的存储器,并通过一个共享存储器与其他处理器通讯. PRAM 可根据存储器的读写方式分类:CRCW (concurrent read concurrent write) PRAM,CREW (concurrent read exclusive write) PRAM 和 EREW (exclusive read exclusive write) PRAM. 对于 CRCW,根据存储器冲突的解决办法,还可以进一步分类.[3]

在 EREW 上,已经证明可以用 n 个处理器在 $O(\log n)$ 时间内将 n 个项目排序[4,5],还有将 n 个项目在 $k<n$ 个处理器上排序,并在 $O((n/k+\log n)\log n)$ 时间内完成,对于 $k\leqslant n/\log n$ 来说,这是最佳时间,因为排序的下界是 $O(n\log n)$[6],也有试图去获得简单的并行排序算法的研究,这些算法虽然在最差的情况下不是最佳,但平均来说运行得很快,快速排序算法(Quicksort)就是其中的一个.[7~9]

Martel 和 Gusfield[9]在 CRCW PRAM 上提出了一个并行排序算法,这个算法平均用 $O(\log n)$ 时间将 n 个项目排序,但他们的算法需要 $O(n^3)$ 空间. Chlebus[8]给出了一个类似的算法,把空间的要求从 $O(n^3)$ 减到 $O(n)$. Heidelberger[7]等在一个共享存储器多处理机上提出了一个使用 fetch—and—add 操作的并行快速排序算法.

本文证明可以在 EREW PRAM 上设计快速排序算法. 我们在 EREW PRAM 上给出一个并行快速排序算法,使用 k 个处理器在平均 $O((n/k+\log n)\log n)$ 时间内和 $O(n)$ 空间内将 n 个项目排序,其时间和处理器数量的乘积,对任何 $k\leqslant n/\log n$,平均来说是 $O(n\log n)$.

---

* 作者管丽,女,1955 年生,讲师,主要研究领域为计算机应用,数据库.
本文通讯联系人:管丽,北京 100875,北京师范大学
本文 1995—06—23 收到修改稿

1 在 EREW PRAM 上的并行快速排序算法

给定 n 个项目的一个数组,$a[1...n]$,传统的快速排序算法有以下几个步骤:

(1) 在数组 a 上选一个项目 s 作为分点.

(2) 将所有小于或等于 s 的项目移到 s 的左边,所有大于 s 的项目移到 s 的右边,现在 s 是在它最终位置上,s 左边的所有项目组成一个子数组,s 右边的所有项目也组成一个子数组.

(3) 在这 2 个子数组上重复步骤(1)~(3).

我们的并行快速排序算法也遵循这几个步骤,令 $P_1,P_2,...,P_k$ 为 k 个处理器,在这个算法中,我们要用到 2 个子程序:

给定一个数组 $x[1...n]$,第 1 个子程序 PREFIX,计算 n 个前缀之和:

$$Si = x[1] + x[2] + \cdots + x[i] = \sum_{j=1}^{i} x[j], \qquad i = 1,2,...,n$$

在 EREW PRAM 上,有一个简单的计算前缀之和的算法,用时间 $O(\log n)$,使用 $O(n)$ 个处理器.[3]实际上,对任何 $k \leqslant n/\log n$,这个算法可以在 $O(n/k)$ 时间内用 k 个处理器完成,常常一组处理器只需在一个数组的一部分项目上计算前缀之和,所以我们定义 $PREFIX(x,L,R)$ 为一子程序:即处理器 $P_L,P_{L+1},\cdots,P_R$ 在 $x[L...R]$ 上计算前缀之和,计算结果存在 $x[L...R]$ 上.

另一个将要用到的子程序是传播(BROADCAST),$BROADCAST(v,L,R)$ 将一个值 v 传给所有下标在 L 和 R 之间的处理器. $BROADCAST(v,L,R)$ 可以用 $PREFIX(tem,L,R)$ 来完成,这里 tem 是一个临时数组,先将 v 赋于 $tem[L]$,其它项目为 0,然后计算 tem 的前缀之和 $PREFIX(tem,L,R)$,其结果是将 v 传给 $P_L,P_{L+1},\cdots,P_R$.

并行快速排序算法. 给定 k 个处理器,$P_1,P_2,...,P_k$,和 n 个项目 $a[1...n]$,每个处理器在初始时被分以 n/k 个项目,称为一块(最后一块可能少于 $[n/k]$ 个项目,但不影响算法),块的大小以后会变化,每个处理器需要 4 个临时变量:L_p 和 R_p 记录一组处理器的第 1 个和最后 1 个,L_b 和 R_b 记录一组项目的左右界. 此外,3 个数组 $S[1...k]$,$B[1...k]$ 和 $b[1...n]$ 用作临时存储.

(1) 初始化:每个处理器作如下操作:

$L_p=1, R_p=k$

$L_b=1, R_b=n$

(2) P_{Lp} 选一个分点 s

(3) $BROADCAST(s,L_p,R_p)$

(4) 划分:

(a) 查块中的项目个数

① 处理器 P_i 计算块 i 中小于或等于 s 的项目个数,将此数存于 $S[i]$,类似地,将块中大于 s 的项目个数存于 $B[i]$

② $PREFIX(S,L_p,R_p)$

③ $PREFIX(B,L_p,R_p)$

④ $BROADCAST(S[R_p],L_p,R_p)$

(b) 移动数据

如果 c 是块 i 中第 j 个小于或等于 s 的项目,则

P_i 将 c 送到 $b[L_b+S[i-1]+j-1]$

如果 c 是块 i 中第 k 个大于 s 的项目,则

P_i 将 c 送到 $b[L_b+S[R_p]+B[i-1]+k-1]$

(c) 调整处理器边界:对每个处理器 i,如果 $S[R_p]$在块 T 中,则 $S[R_p]$将块 T 分成两半,如果左半小于右半则做①,否则做②

① 如果 $i<T$,则$(L_p,R_p)=(L_p,T-1)$;否则$(L_p,R_p)=(T,R_p)$

② 如果 i<T,则$(L_p,R_p)=(L_p,T)$;否则$(L_p,R_p)=(T+1,R_p)$

(d) 调整块边界:

如果 $S[R_p]$恰是某个块的边界,转下一步,否则,$S[R_p]$在一个块当中,将该块分成两半,令该块保留较大的那半,将较小的那半合并到与其邻近的那块,如两半相等,则可以任意分

(e) 确定 2 个子数组的边界

如一个处理器在 $S[R_p]$左边　$(L_b,R_b)=(L_b,L_b+S[R_b]-1)$

如一个处理器在 $S[R_p]$右边　$(L_b,R_b)=(L_b+S[R_b],R_b)$

(5) 如果 $L_p=R_p$,则该处理器将 $b[L_b,R_b]$排序,然后停机

(6) 递归应用步骤(2)~(5),相应的交换数组 a 和 b 的作用

图 1 给出了一个例子.

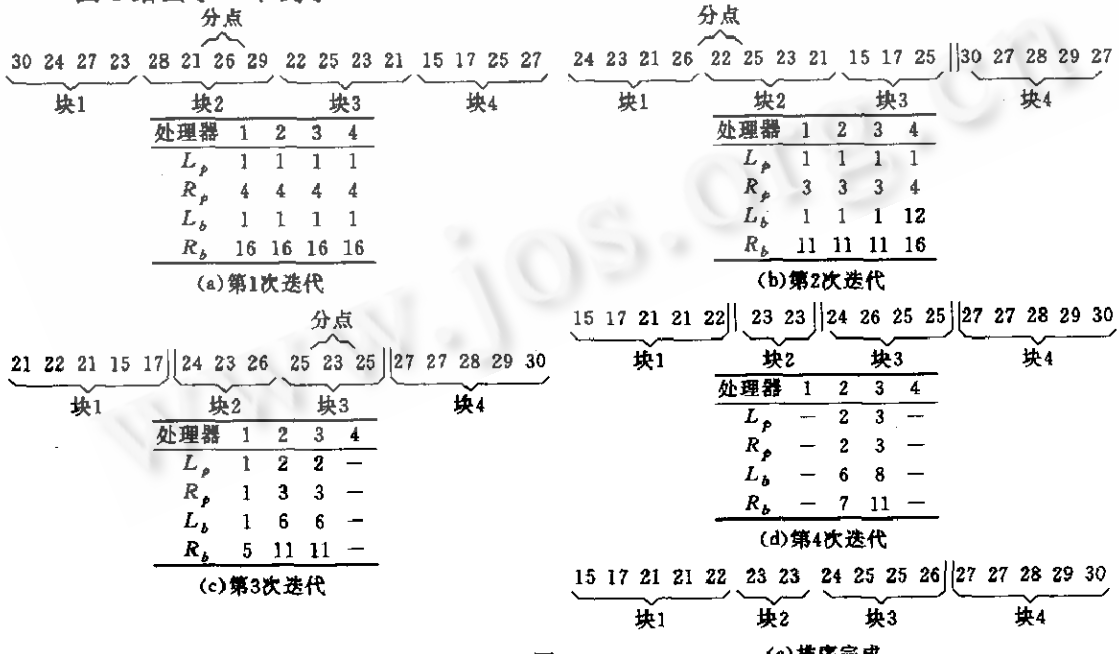

图1

2 时间和空间分析

我们先分析一下每一次迭代的时间,这取决于块的大小和处理器的个数,因为在整个算法运行过程中,每个处理器最多扩展 2 次块的边界(1 次向左,1 次向右),且每块的大小最多只增加 $1/2(n/k)$,所以每块的大小从不会超过 $2(n/k)$.

步骤(1)、(2)需 $O(1)$时间,步骤(3)需 $O(\log k)$,步骤(4)需 $O(\log k+n/k)$,这里第 1 项来自 PREFIX 和 BROADCAST,第 2 项来自查项目个数和移动数据,步骤(4)的其它操作((c),(d),(e))需 $O(1)$时间,我们注意到每个处理器只运行步骤(5)1 次(可以用任何 1 个最佳顺序排序程序),所以步骤(5)的时间 $O((n/k)\log(n/k))$只需加在整个算法运行时间1 次.

如果我们随机地选择分点,快速排序的平均迭代次数是 $O(\log n)$.[10]所以,我们的并行快速算法的平均运算时间是

$$O((n/k+\log k)\log n+n/k\log(n/k))$$

即

$$O((n/k+\log n)\log n)$$

时间和处理器的乘积,对任何 $k<n/\log n$,平均来说是 $O(n\log n)$,算法对空间的要求是 $O(n)$.

3 结 论

在 EREW PRAM 上,给定 k 个处理器,我们提出了一个并行快速排序算法,将 n 个项目排序,其时间和处理器个数的乘积,对任何 $k\leqslant n/\log n$,平均来说是 $O(n\log n)$. 是否能在 EREW 上用 n 个处理器将 n 个项目在 $O(\log n)$时间排序,将是一个很有趣的问题.

参考文献

1 Akl S G. Parallel sorting algorithms. Orlando: Academic Press, FL, 1985.

2 Bitton D, Dewitt D J, Hsiao D K *et al*. A taxonomy of parallel sorting. Computing Surveys, 1984, **16**: 287～318.

3 Karp R M, Ramachandran V. A survey of parallel algorithms for shared—memory machines. In: Leeuwen J Van ed. Handbook of Theoretical Computer Science, North Holland Amsterdam, 1990.

4 Ajtai M, Komlos J, Szemeredi E. An $O(n\log n)$ sorting network. Proc. 15th ACM Symp. on Theory of Computing, 1983. 1～9.

5 Cole R. Parallel merge sort. SIAM J. on Computing, 1988, **17**: 770～785.

6 Guan X, Langston M A. Time—space optimal parallel merging and sorting. IEEE Transactions on Computers, 1991, **40**: 596～602.

7 Heidelberger P, Norton A, Robinson J T. Parallel quicksort using fetch—and—add. IEEE Transactions on Computers, 1990, **39**: 133～138.

8 Chlebus B S. Parallel quicksort. Journal of Parallel and Distributed Computing, 1991, **11**: 332～337.

9 Martel C U, Gusfield D. A fast parallel quicksort algorithm. Information Processing Letter, 1989, **30**: 97～102.

10 Robson J M. The height of binary search trees. Australian Computer Journal, 1979, **11**: 151～153.

PARALLEL QUICKSORT WITHOUT MEMORY CONFLICTS

Guan Li

(*Beijing Normal University　Beijing*　100875)

Abstract　A parallel quicksort algorithm is given and which, given an EREW PRAM with k processors, sorts n items in expected $O((n/k+\log n)\log n)$ time, and thus the product of time and number of processors is $O(n\log n)$ on the average for any value of $k\leqslant n/(\log n)$.

Key words　Parallel algorithm, sorting algorithm, parallel quicksort algorithm.