最高位优先基数排序的一种实现方法[20]

黄竞伟

(武汉大学软件工程国家重点实验室,湖北 武汉 430072)

摘要 给出了最高位优先基数排序算法的一种实现算法,与其它的实现方法相比,该方法易于扩展为异步并行基数排序算法.

关键词 排序;算法;算法分析

An Implementation Method of MSD Radix Sort

HUANG Jingwei

(State Key Lab. of Software Engineering, Wuhan University, Wuhan 430072)

Abstract An implementation method of MSD radix sort is presented. Compared to other implementation methods of radix sort, it is easy to be extended to asynchronous parallel radix sort algorithm.

Keywords sorting; algorithm; algorithms analysis

基数排序是一种常用的排序方法,基数排序将关键字看作是以某个正整数 r 为基的数,然后依次按关键字的各个字位分别对记录进行排序. 通常有两种方法实现基数排序:最高位优先(简称 MSD法)和最低位优先(简称 LSD法). LSD法易于实现,一般在数据结构教科书[1,2]中讨论较多. 若按 MSD法必须将待排序序列分割为若干子序列,然后对各子序列再分别排序,实现起来较为困难,文献[3]推广了 LSD链式基数排序算法,给出了 MSD法的一种实现方法,但该方法不易于扩展为并行算法. 在本文中,我们给出了 MSD法的一种实现方法,该方法易于扩展为异步并行基数排序算法.

1 MSD法的一种实现方法

设有 n 个记录的序列 $\{R_1, R_2, \cdots, R_n\}$,每个记录 R_i 的关键字 k_i 是一个 d 位 r 进制数,即设

$$k_i = (k_i^1, k_i^2, \cdots, k_i^d),\ 0 \leqslant k_i^j \leqslant r-1,\ 1 \leqslant j \leqslant d$$

其中关键字的第 1位称为最高位,第 d 位称为最低位,k_i^1 称为最高位关键字,k_i^d 称为最低位关键字,按 MSD法对序列 $\{R_1, R_2, \cdots, R_n\}$ 排序的思想是首先按最高位关键字对记录进行排序,结果可以得到按最高位关键字排好序的若干记录的子序列,每一子序列中记录的最高位关键字相同,第一个子序列中记录的最高位关键字最小,最后一个子序列中记录的最高位关键字最大,再用同样的方式分别对这若干子序列中的每一子序列中的记录按关键字 k_i^2 进行排序,将其再分为若干子序列,这时每一子序列中记录关键字 k_i^1, k_i^2 都相同,然后再用同样的方式分别对这若干子序列中的每一子序列中的记录按关键字位 k_i^3 进行排序,这样继续下去,最后分别对每一子序列中的记录按关键字 k_i^d 排序,再将所有的子序列依次联接在一起就得到排好序的序列.

例 用 MSD法对下列关键字序列排序

$$\{278, 109, 063, 064, 930, 589, 184, 505, 269, 008, 083\}$$

[20] 收稿日期:1998-03-17

按最高位排序后,原关键字序列分解为下列子序列

{063, 064, 008, 083}, {109, 184}, {278, 269}, {589, 505}, {930}

按次高位分别对上述各子序列排序后得到

{008}, {063, 064}, {083}, {109}, {184}, {269}, {278}, {505}, {589}, {930}

按最低位分别对上述各子序列排序后得到

{008}, {063}, {064}, {083}, {109}, {184}, {269}, {278}, {505}, {589}, {930}

最后再将所有的子序列依次联接在一起就得到排好序的序列

{008, 063, 064, 083, 109, 184, 269, 278, 505, 589, 930}

下面我们描述 MSD法的一种易于并行化的实现算法.

从上面的例子可以看出,MSD法实际上是按某个关键字位对当前得到的各个子序列分别进行排序,而且各个子序列的排序过程可以独立地进行. 假设待排序的 n 个记录存放在数组 $R[1,\cdots,n]$中,排序过程中出现的各个子序列仍然依次存放在数组 $R[1,\cdots,n]$中. 为了对某个子序列排序,我们需要知道该子序列在数组 $R[1,\cdots,n]$中的起始位置,该子序列的长度即子序列中记录的个数以及对该子序列进行排序时所用的关键字位. 为此我们用一个循环队列 $Q[0,\cdots,n-1]$中存放各个子序列在数组 $R[1,\cdots,n]$中的起始位置,子序列的长度和对该子序列进行排序时所用的关键字位. 在开始排序时,仅有一个子序列即给定的记录序列,因而初始时队列 Q 中仅存放给定记录序列的起始位置 1,长度 n,排序关键字位 1.

在按关键字位 j 对一个子序列排序时,我们采用下列方法: 首先计数 $k_i^j = t$ $(0 \leqslant t \leqslant r-1)$的个数,将其记录在数组 NUM[0,…,r-1]的第 t 个单元 NUM[t]中,并计算关键字位 $k_i^j = t$ 的第一个记录在该子序列中的起始位置,并用数组 POS[0,…,r-1]的第 t 个单元 POS[t]记录. 若该子序列在数组 $R[1,\cdots,n]$中的起始位置为 Start,显然有

$$POS[0] = Start,$$

$$POS[t] = POS[t-1] + NUM[t-1], \quad (0 \leqslant t \leqslant r-1)$$

然后依次扫描该子序列,若当前记录 R_i 的第 j 位关键字 $k_i^j = t$,则将 R_i 放入辅助数组 $B[1,\cdots,n]$的第 POS[t]个位置,然后置 POS[t] = POS[t] + 1. 待按关键字位 j 对各个子序列排好序后,再将数组 B 写回数组 R,这时该子序列已按第 j 位排好序了. 对上述方法的一个改进是,在按关键字位 j 对一个子序列排序时,首先检查该子序列的第 j 位关键字是否全相同,若全相同,则在 $j < d$ 时继续对该子序列的第 $j+1$ 位关键字进行排序,否则用上述方法对该子序列的第 j 位关键字排序.

在下面的算法中,我们对记录使用了如下的类 Pascal说明:

```
Type Node = Record
              Key= Array[1,…,d] of 0,…,r-1;
              Otherfields;
            End;
     ArrayType= Array[1,…,n] of Node;
```

所用循环队列 Q 的说明如下:

```
Type QueueNode= Record
                  Start: 1,…,n;
                  Len: 1,…,n;
                  KeyPos: 1,…,d;
                End;
     QueueType= Array[0,…,n-1] of QueueNode;
```

在下面的算法中,我们使用了如下队列操作:

InitQueue(Q): 初始化队列 Q;

EnQueue(Q, Start, Len, KeyPos): 将数组 $R[1,\cdots,n]$中的起始地址为 Start,长度为 Len且当前排序

关键字位为 KeyPos的子序列入队.

DeQueue(Q, Start, Len, KeyPos): 将数组 $R[1,\cdots,n]$中的起始地址为 Start,长度为 Len且当前排序关键字位为 KeyPos的子序列出队.

Empty(Q): 判别队列 Q是否为空;

*MSD*法的一个实现方法由下面的算法 1形式地给出.

算法 1 *MSD*基数排序算法:

```
Procedure Radix-Sort (Var R: Array Type; n: integer);
begin
1)  initQue ( Q ); enqueue ( Q, 1,n, 1 );
2)  While Not empty( Q ) Do
3)  Dequeue ( Q , Start, Len, KeyPos); Equal← True;
4)  While Equal Do
5)  i← Start+ 1;
6)  While ( i <= Satrt-Len+ 1) and (Equal) Do
7)  If R[Start]. Key[KeyPos]= R[i] . Key [KeyPos]
8)    Then i← i+ 1
9)    Else Equal← False;
10) If Equal Then
11)   If KeyPos < d Then KeyPos← KeyPos+ 1
12)     Else Equal← False;
13) EndWhile
14) If KeyPos < d Then
15)   For j = 0 To r - 1 Do
16)    Num [j]← 0; Pos [j]← 0;
17)   End For
18)   For j = Start To Start+ Len- 1 Do
19)    Num [R[j] . Key[KeyPos]]← Num [R[j] . Key [KeyPos] ]+ 1;
20)   Tpos← 0;
21)   While Num[Tpos]= 0 Do Tpos← Tpos+ 1;
22)   Pos[Tpos]← Start;
23)   If ( Num[ Tpos]> 1) and ( KeyPos < d ) Then
24)     Enqueue ( Q , Start, Num [Tpos], KeyPos+ 1);
25)   For j = Tpos+ 1 To r - 1 Do
26)    Pos[ j ]← Pos[ j - 1 ]+ Num[j - 1 ];
27)    If (Num[ j ]> 1) and ( KeyPos < d )
28)      Then Enqueue ( Q, Pos[j ], Num[j ], KeyPos+ 1);
29)   End For;
30)   For j = Start To Start+ Len- 1 Do
31)   B[Pos[ R[j ]. Key[KeyPos]]]← R[j ];
32)    Pos[ R[j ]. Key [KeyPos]]← Pos[ R[j ]. Key [KeyPos] ]+ 1;
33)   End For;
34)   For j = Start To Start+ Len- 1 Do R[j]← B[j ];
35) EndIf;
```

36) EndWhile;
37) End;

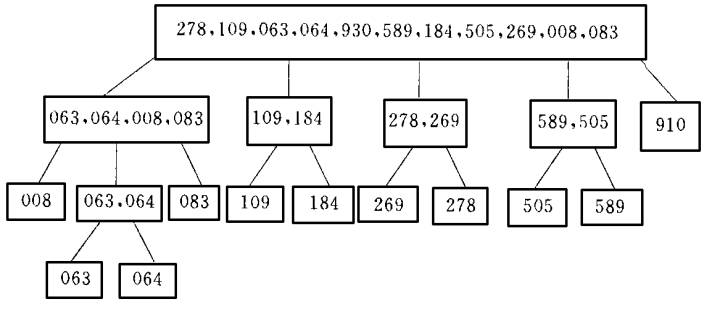
2 算法的正确性证明及算法分析

定理 1 算法 1正确地对 n 个记录排序.

证明 我们如下证明算法 1的正确性,对于队列 Q中的任何子序列 S,若其排序关键字位为 j,则该子序列已按关键字位 1,2,… ,j- 1正确地排序,且所有不在队列 Q子序列中的记录已被正确地存放在数组 R中,这样当 Q为空时,所有的记录已被正确地存放在数组 R中了.

初始时,队列 Q中仅含有一个子序列即给定的待排序的记录序列,其排序关键字位为 1,显然该子序列已按关键字位 0正确地排序. 在执行 While循环时,若一个子序列 S由语句 3)出队时,其排序关键字位为 j,我们首先看该子序列的第 j位关键字是否都相同,若都相同,则该子序列已按第 j位排好序了,当 j< d时,我们再按第 j+ 1位关键字对该子序列排序,当 j= d时,则该子序列已排好序了,这由步骤 4)~ 13)完成,若不完全相同,则经过步骤 15)~ 29)后,该子序列被分为若干子序列 $S_1, S_2, \cdots, S_k$ ($0 \leqslant k \leqslant r-1$),其中 S_i中记录的第 1,2,… ,j位关键字相同,且 S_i中记录的第 j位关键字小于 S_{i+1}中记录的第 j位关键字 ($1 \leqslant i \leqslant k-1$),步骤 30)~ 34)将子序列 $S_1, S_2, \cdots, S_k$ 按第 j位关键字正确地存放在数组 R中了. 若某子序列仅有一个记录,显然该记录在数组 R中的位置已是该记录在排好序的序列中的最后位置,若某子序列有多于一个的记录,则步骤 24)~ 28)将该子序列放入队列 Q中,下次将按第 j+ 1位关键字对该子序列排序,当所有子序列都已按关键字位 d排好序后,这时队列 Q为空,且所有记录都已被正确地存放在数组 R中了,这就归纳地证明了定理 1.

为了分析算法 1的时间复杂度,首先估计放入队列 Q中子序列的个数. 我们可以用一个 r叉树描述子序列的产生过程. 例如对于关键字序列 {278, 109, 063, 064, 930, 589, 184, 505, 269, 008, 083},我们有下列 r叉树:



每个入队的子序列可以用 r叉树中的内结点表示,故要估计入队的子序列的个数,只要估计内结点的个数即可.

引理 入队子序列的个数不超过 n- 1.

证明 由算法 1知,对于一个子序列,仅当对当前关键字位该子序列有两个不同的关键字时,该子序列才入队,故 r叉树中每个内结点至少有两个儿子.

设 r叉树中度为 0, 1,… ,r的结点个数分别为 $n_0, n_1, \cdots, n_r$,那么有

$$n_0 + n_1 + \cdots + n_r = n_1 + 2 * n_2 + \cdots + r * n_r + 1, n_0 = n$$

故有

$$n_2 + 2 * n_3 + \cdots + (r-1) * n_r = n_0 - 1,$$

于是内结点的个数

$$n_2 + n_3 + \cdots + n_r \leqslant n_2 + 2 * n_3 + \cdots + (r-1) * n_r = n_0 - 1 = n - 1$$

(下转第 111页)

[4] 钟安环 . 生物学引论 . 北京: 中国人民大学出版社 , 1983
[5] 许国志等编著 . 系统科学大辞典 . P622"要素" ,昆明: 云南科技出版社 , 1994
[6] 张颖清 . 全息生物学 (上) . 北京: 高等教育出版社 , 1989
[7] 梁俊雄 . 全息理论的数学基础 (概述) . (云南省第二届青年学术年会论文集) ,昆明: 云南科技出版社 , 1996
[8] 尚玉昌等编著 . 普通生态学 (上册) . 北京: 北京大学出版社 , 1992
[10] 邹珊刚等编著 . 系统科学 . 上海: 上海人民出版社 , 1987
[11] [英]肯尼思° 法尔科内著 ,曾文曲等译 . 分形几何—— 数学基础及其应用 . 沈阳: 东北工学院出版社 , 1991
[12] 张济忠 . 分形学 . 北京: 清华大学出版社 , 1995
[13] 敖力布 ,林鸿溢主编 . 分形学导论 . 呼和浩特: 内蒙古人民出版社 , 1996
[14] 梁俊雄 . 全息理论的数学基础 (Ⅰ)—— 全息胚的枝形结构 . 武汉: 华中师范大学学报 (专辑) , 1996 11～ 15

(上接第 105页)

定理 **2** 算法 1的时间复杂度为 $O((d+r)n)$.

证明　算法 1的时间复杂度由语句 7) , 16) , 31)所决定 . 在每次 While循环中 ,当按关键字位 $j(1\leqslant j\leqslant d)$对记录进行排序时 ,语句 7) , 30)最多执行 n次 ,故语句 7) , 30)共执行 $O(dn)$次 . 对 Q中的每个子序列 ,语句 15)执行 r次 ,由引理知 ,最多有 $n-1$个子序列入队 ,因而语句 16)最多执行 rn次 ,故算法的时间复杂度 $O((d+r)n)$.

3 结语

在本文中 ,我们给出了最高位优先基数排序算法的一个实现方法 ,时间复杂度与最低位优先基数排序算法一样 ,同为 $O((d+r)n)$,关于将上述算法扩展为异步并行基数排序算法的研究工作 ,我们将另文讨论 .

参 考 文 献

[1] 严蔚敏 ,吴伟民 . 数据结构 . 北京: 清华大学出版社 (第二版) , 1992
[2] 施伯乐 ,蔡子经 ,孟佩琴 ,张乃玲 . 数据结构 . 上海: 复旦大学 , 1988
[3] Knuth D E. The Art of Programming. Sorting and Searching. 1973, 3