图象处理中边界转换的并行算法及其实现*

杨 勃 陈 虎 陈国良

(中国科学技术大学计算机系 合肥 230027)

摘要 本文提出了一种把图象中边界转换成区域四分树的并行方法.该方法基于 MIMD 模型,并在曙光 1000 上实际运行.整个算法用 P 个处理器可以在时间 $O((B\times\log B)/P)$ 内完成其中 B 是循环代码长度.该算法可应用于图象处理、计算机图形学、模式识别等领域.

关键词 四分树,Morton 序列,Jordan 曲线,循环代码,快速排序.

中图法分类号 TP391.41

任何图象都可以通过一个 N×N 象素阵列来表示.每 P 位图象在不编码情况下,则需要 $P\times N\times N$ 位来表示.实际上,利用邻近的象素常是相同的和一维与二维图象的一致性等特点,用合适的编码方式,可以用远小于 $P\times N\times N$ 位来表示图象.[1]用四分树对一个图象编码,就是利用二维图象的连续性来递归地把 $2^n\times 2^n$ 图象阵列划分成 4 个相等的象限[2],如果这个阵列的所有点灰度不同,这个图象将进一步划分成子四分象限、子子四分象限……,直到每块是相同的.用指针型树结构来表示四分树需要大量的空间来存储结点和这些结点指向它们子结点的指针.[3]我们采用无指针的四分树,即线性四分树,以便节省大量空间.由此可见,四分树有两个主要的应用领域:①由于许多图象的内部一致性,可用于图象的有效压缩存储;②由于四分树的特性,在图象不同处理中可有助于快速获得我们感兴趣的区域.

本文描述了把循环代码转换成线性四分树的算法[4],同时将算法在曙光 1000 并行机上实现.这对图象的表示和处理是很有用的.以前的几种转换方法存在着以下问题.第 1 种算法[5],首先建立一个四分树,每个边界点从循环代码分离出来作为一个 BLACK 结点插入,树的构造沿边界点的处理过程中完成,当处理循环代码时采用邻近寻找操作.第 2 个阶段是遍历四分树,用循环代码转化成指针四分树来表示,所占空间很大;第 2 种算法[6],把循环代码转换成线性四分树.这种线性四分树用原始树的叶子结点构成的表来取代指针数结构,表中结果记录如果被 MORTON 代码排序,则按照这个次序,与中序遍历同一图象的指针四分树的相应块是一致的.我们指的线性四分树,其结点表包括全部叶子结点,但没有用 MORTON 代码排序,所以每个边界点均包含信息.这样,初始代码通过一个递归程序,把该表分裂为 4 部分,依靠每点所在图象的那一象限,然后对每一子表再运行该递归程序,最终依据相点所在图象的子象限,对 $2^n\times 2^n$ 图象进行 n 次划分,而得到的结果是非排序表.这种方法没有在划分同时的排序,重复运算较多,而且在处理较大的图象时,显得尤为困难;第 3 种方法[7],采用路径平衡方法,即把边界表示看成两个一维目标:一个是水平的,另一个是垂直的,来判定多边形在四分树中的位置,以减少邻近搜索操作的总花费.这些边界点可以插入到四分树中,区域内部能填充在与 1 个四分树点的个数成比例的时间内.但它有个致命的弱点,那就是为了获得线性时间复杂度,多边形区域表示必须在树内某一优化位置开始,这种约束是不可忍受的.本文提出的算法是把边界表示转化成 MORTON 代码序列的线性四分树.这种算法克服了以往算法的缺点,对于大图象的表示与处理,如 1024×1024,2048×2048 等更加有效.

本文第 1 节描述了算法的基本思想;第 2、3 节讨论了算法的并行实现及分析;第 4 节是算法在曙光 1000 上试验的结果;最后是小结.

1 算法的基本思想

我们的算法把边界表示转化成 MORTON 代码序列的线性四分树,边界表示可以是循环代码或多边形周长的向

---

* 本文研究得到国家教委博士点基金资助.
作者杨勃,1962 年生,博士生,工程师,主要研究领域为图象处理、计算机视觉、高性能并行计算的应用.陈虎,1973 年生,硕士生,主要研究领域为并行算法,高性能并行计算,图象处理.陈国良,1938 年生,教授,博士导师,主要研究领域为并行算法,高性能并行计算,计算智能.

本文通讯联系人:杨勃,合肥 230027,中国科学技术大学计算机系

本文 1996-01-03 收到原稿,1997-06-09 收到修改稿

量,所要满足的条件是必须能转化为边界点表.多边形内部结点当转化进程结束后标为 BLACK;多边形外部节点标为 WHITE.如图 1 中,(a)为图象的象素点值;(b)为转换后的图象,标号为四分树结点;(c)是图象的四分树结构.

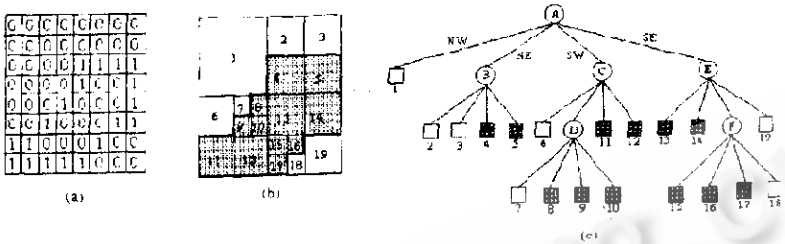

图 1 边界点到四分树的转换

算法的第 1 步需遍历边界产生边界点表;第 2 步用 MORTON 代码排序这个边界点表;第 3 步是多边形填充阶段,即把已排序的 MORTON 代码转换成线性四分树.

1.1 由循环代码到边界点表

循环代码是描述边界的一种压缩表示.边界代码的起始点和边界代码的走向包括了边界的全部信息,由此可重现边界.本文用到了边界代码其中的一种表示,包括边界的起始和方向的定向代码.首先转化为边界点表,这个边界点表必须能构成 JORDON 曲线.JORDON 曲线严格描述如下:①每个多边形必须是封闭的(同一点开始,同一点结束).②多边形不能是自相交的,也不能是两个多边形在图象上相交.

边界点表中的记录包括每个边界点的 X,Y 坐标,一个边界代码表示该点的哪一边邻近 WHITE 点(点的哪一边接触循环代码边界).边界代码域在算法的第 3 阶段的更新色表中会影响到下一邻近点的颜色.这就是所采用的期望法.边界点表的初始点坐标由循环代码的起始坐标决定.最后,还要判定是否满足 JORDON 曲线的条件.

转换方法按图 2 的示例,共有 3 种情况来判定是否有新的边界点产生或只修改边界域.由于曲线可分为八连接的和四连接的,边界点表也可分为八连接的和四连接的.图 2(a)表示,增加新的边界点记录和它相应的边界代码值.图 2(b)表示,不增加边界点记录,只增加边界代码值.图 2(c)表示,按照四连接曲线,需增加两个边界点记录;按照八连接曲线,需增加一个边界点记录.

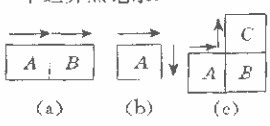

图2 循环代码到边界点表

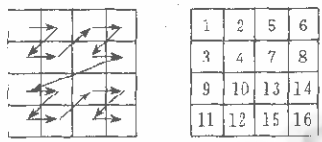

图3 MORTON序列

图4 边界代码

1.2 边界点表到 MORTON 代码序列

当边界点表形成后,排序为 MORTON 序列.如图 3 所示,MORTON 序列是与递归遍历四分树的顺序一致的代码,图中的数字顺序即是 MORTON 序列的顺序.这一过程可采用各种不同的方法,我们这里采用了快速排序方法.

1.3 MORTON 代码到四分树

第 3 部分描述了从 MORTON 序列的边界点表到四分树的转换过程.当产生了初始边界点表后,每一对应点有一个边界代码,这个边界代码描述哪个边界邻近 WHITE 点.把代码看成四象限映射,0,1,2,3 对应方向 N,E,S,W.四值可通过计算 N=1,E=2,S=4 和 W=8 生成.如图 4 所示,在区域 SE 角一点有边界代码 2+4=6.注意所有边界点的颜色为 BLACK,因为定义边界点在循环代码定义的多边形内部.

当边界点表排序成 Morton 代码序列后,在模拟指针四分树遍历时,通过处理 Morton 代码点表生成完全线性四分树.为帮助遍历,信息存储在色表中.这个表是二维阵列,n 行对应于 $2^n\times 2^n$ 图象,即一行代表树的一层(除了 n 层的根),每行有 4 列,对应四分限 NW,NE,SW,SE.因此,表中每一项对应于树中一定层的一个四分限.色表中每一项中的颜色表示四分树上对应的结点的颜色.

遍历开始时,设结点表当前层为 n,对应于指针树的根访问,遍历初始位置是(0,0),即最左上角.在遍历的每一阶段,我们保持树的当前层,树内的当前位置(x,y),描述当前点的 3 个角点的 3 个角值(除了 SE 角).例如,如果忽略当前结点的实际层,点(0,0)是整个图象的 NW 角,而点(4,0)是 4×4 点结点的 NW 角.我们保留这个角信息是为了允许不断更新色表,如果在这个结点的边界代码中有边界表示,则可以依据边界代码更新色表的相应项.在遍历过程中,当前结点的角值能随时被其父亲的角值替代.

处理点表是收集那些位于每一对边界点之间的点的碎片问题.当一结点在一碎片内要输出时,色表对每一结点更新.一个碎片内第1个结点是边界点本身,因为这个边界点(一定是NW角)可以有E或S边的循环代码边界.只要适合,这个边界点会与四分组归并,并且当前点位置增加到Morton代码的下一位置.当碎片中下一结点的NW角不包括下一边界点时,它是当前点位置最大四分树块.当这一结点输出时,下一点位置轮流计算——这将是沿处理结点的Morton序列的下一点.当当前点位置与边界结点表中下一边界点位置相同时,当前碎片处理完,开始处理下一碎片,直到边界点表全被处理.模拟树遍历的目的是允许线性时间内计算构成当前碎片的结点.在处理完最后的边界点之后,最后的碎片包含图象剩余部分,它的颜色就是WHITE.

在本阶段,一边遍历边界点表,一边模拟指针四分树遍历.如果当前层的点不适合并入当前片,算法递归处理该结点的4个孩子.对每个孩子,计算NW角坐标,即最大角值.一个最大角值是最大块的层,这个块是在对应角内该点的块.结点N最大的东邻近是最大SW角值.如果当前结点的块在四分树遍历中适合并入当前碎片,这就决定了当前结点的颜色,再调用结点合并过程.如果当前结点是边界点,它的颜色是BLACK.如果不是,它的颜色由色表相应项决定.色表项检查在该层和当前结点的NW点的最大角值的象限,在结点颜色决定后,该颜色可用于更新色表.\; 如果当前结点是它父亲的NW,NE或SW孩子,色表更新.对SE孩子,不需要更新色表,因为其它结点以后会更新这些表项.色表更新如下:对当前结点,计算最大东邻近和南邻近的大小,它们从当前结点的最大角值获得.如果当前结点是边界点,这点的边界代码决定了更新色表时的颜色.例如,如果该点存储了沿东边沿的一边界,那么当前点的最大象限东方向置为WHITE,这可通过设定色表中最大东邻近的表项(层和象限)来完成.如果当前结点不是边界点,那么邻近结点接受结点的值.

2　并行算法的实现

基于以上算法思想的分析,我们采用并行计算加快处理.对$2^n\times2^n$图象,用$2^i\times2^i$处理器($i=0,1,\ldots,n$)均分,每个处理器处理$2^{n-i}\times2^{n-i}$的图象.[8,9]

2.1　并行算法实现步骤

第1步:采取分割数据的方法,即分割处理循环代码,使其在各个处理器上生成自身的满足JORDAN曲线的边界点MORTON序列.

第2步:调用模拟指针四分树遍历过程,每个处理器处理自己的$2^{n-i}\times2^{n-i}$图象.

第3步:在各个处理器处理完各自的图象后,连接各四分树.

并行算法的关键是第1步——分割图象可有以下两种情况.情况1如图5(a)所示,处理器分割曲线,截得曲线(A,B),按构成边界点的生成方法,多边形(A,B,C,D)应是所求的区域;情况2如图5(b)所示,处理器处理的图象内不含边界点,可按曲线经过分割块上方的奇偶性判断该块的颜色.若为偶,该块为WHITE,若为奇,该块为BLACK.由此算法的时间将大大减少.但由于图象的多边形填充过程需满足JORDAN曲线,为了分割图象,图象分割块的曲线也是JORDAN曲线.

(a)　　(b)

图5　分割图象

2.2　并行算法

输入:已排序的边界点表codelist

输出:四分树quadtree

```
begin
(1) for all p=0~2^i×2^i-1 pardo
        algorithm(p)
    endfor
(2) p个处理器采用四分树法合并四分树
end

procedure algorithm(p)
begin
```

```
(1) while i<MAXLIST do
        If 边界点在该处理器截取的方块内 then
          codelt[j]=codelist[i]
          j=j+1
          i=i+1
        endif
(2) endwhile
    if(j>0) then            /*情况 1*/
      CONNECT(j-1)
      START_TRAVERSE(x,y)
      else            /*情况 2*/
      if JUDGE(x,y) then
      输出全 BLACK 结点
      else
      输出全 WHITE 结点
      endif
    endif
end
```

情况 1:联接边界点的算法.

```
procudure CONNECT(i)
begin
(1) for j=0 to i do
      if 边界点与处理器处理的图象的边界相交 then
        lt[k]=j
        k=k+1
        if 出分割块边界点邻近入分割块边界点 then
          lt[k-2]=lt[k-1]
          k=k-1
        endif
      endif
    endfor
(2) while k>0 do
    (2.1) 根据出分割块边界点的边界代码来判断方向(incx,incy 的正负)
    (2.2) if incx>0 or incy>0 then
          while 增加的边界点未到达图象角或入分割块边界点 do
            i=i+1
            codelt[i]的坐标等于 codelt[lt[k]]的坐标加上 X,Y 的增量(incx 和 incy)
            边界代码更新
          endwhile
          incx=0
          incy=0
    (2.3) if 不是入分割块边界点 then
            lt[k]=i
            k=k+1
          endif
          k=k-1
          endwhile
end
```

情况 2:方块内无边界点的算法.

```
function JUDGE(x,y)
begin
(1) for j=0 to MAXLIST do
      if 经过 A 点上方的边界点的边界代码为 N 或 S then
        i=i+1
      endif
    endfor
(2) if i一偶数 then
```

```
        return FALSE
        else return TRUE
    endif
end
```

3 分　析

Samet[5]算法实际是把 B 边界点插入第 n 层四分树,Atkinson 等人算法[6]通过长为 B 的结点表完成 n 次传递,而 Mark 和 Abel[7]完成了邻近查找操作在四分树 $O(B)$ 个结点内 $O(n)$ 次操作,他们的算法都是 $O(n\times B)$. 因此所有的早期算法在他们的时间复杂度内有深度因子 n.

我们的算法包含 3 个阶段:第 1 阶段是循环代码处理(或边界重表示),生成多边形边界点的线性四分树结点表. 对于长为 B 的边界点的表,它的时间复杂度是 $O(B)$;第 2 阶段是 MORTON 代码排序[10],需时间 $O(B\times\log B)$. 第 2 阶段输出的是边界点结点的有序表;第 3 阶段是 MORTON 序列到四分树的填充阶段,按 Hunter 理论,四分树中结点个数是直接正比于多边形边界象素 B 加上图象的分解,我们遍历过程(TRAVERSE)模拟指针四分树遍历,每个结点处理一次包含 $O(n+B)$ 个结点. 每个结点(即碎片中每部分)是连续处理的. 因此,第 3 阶段填充的时间复杂度为 $O(n+B)$. 算法的全部花费因此为 $O(B\times\log B)$,由初始排序花费决定.

我们采用并行技术实现以上算法,对第 2、3 阶段进行并行处理. 对 $2^n\times 2^n$ 图象,用 $2^i\times 2^i$ 处理器($i=0,1,\ldots,n$)均分,每个处理器处理 $2^{n-i}\times 2^{n-i}$ 的图象. 对于 P 个处理器,各个处理器并行调用串行过程. 因此,第 2 阶段的时间复杂度为 $O(\frac{B\times\log B}{P})$,第 3 阶段的时间复杂度为 $O(\frac{n\times B}{P})$.

4 实验结果

我们的并行算法是在曙光 1000(DAWN-1000)上用 C 语言实现的. 它是一种基于分布存储结构和消息传送机制的大规模并行计算机系统. 它采用了与 Intel 公司的 paragon 相同的消息传递接口 NX, NX 支持无主(Non-Host)模式和主从(Host-Node)模式两种编程模式. 由于我们的并行算法设计,对 $2^n\times 2^n$ 图象,用 $2^i\times 2^i$ 处理器($i=0,1,\ldots,n$)均分,每个处理器处理 $2^{n-i}\times 2^{n-i}$ 的图象. 所以,我们采用了无主模式. 实验数据见表 1.

表 1　实验数据

边界长度 B	$2^n\times 2^n$ 图象 n.	p 处理机数计算的时间 T(s)					
		$p=1$		$p=4$		$p=16$	
		分割时间	运行时间	分割时间	运行时间	分割时间	运行时间
126	5	0.00	0.10	0.00	0.07	0.00	0.07
254	6	0.00	0.27	0.00	0.13	0.00	0.07
346	7	0.00	0.65	0.00	0.26	0.00	0.13
690	8	0.01	2.69	0.01	1.28	0.00	0.84
1 722	9	0.01	13.78	0.01	4.79	0.00	2.62
3 098	10	0.02	37.52	0.01	17.41	0.01	4.00
13 074	11	0.02	164.34	0.02	51.25	0.01	20.26
26 146	12	0.05	697.74	0.03	211.88	0.02	51.27
52 290	13	0.10	2 954.41	0.06	908.53	0.04	211.92
104 578	14	0.20	12 417.15	0.13	3 805.32	0.12	908.64

从图中数据可知:表中的分割时间复杂度为 $O(B)$;当图象较小时,由于程序的加载时间和分块后相对处理的图象边界数目增大,在图象较大时,其加速比较明显. 对 $2^n\times 2^n$ 图象,当 $n>8$ 时的加速比为 $S_P=\frac{O(B)+t_S}{O(B)+t_P}\approx P$;$S_P$ 可参见图 6.

因为串行时间复杂度 $T_S=O(B+\log B+(n+B))$,所以并行时间复杂度 $T_P=O(B+\frac{B\times\log B+(n+B)}{P})$.

通过以上的实验和分析,整个转换算法的时间复杂度是由第 2 阶段决定的. 因此,并行算法的总的时间复杂度为 $O(\frac{B\times\log B}{P})$.

5 结　论

算法达到了 3 个目标:第 1,它实现了边界代码到线性四分树的转化;第 2,它把转化过程分为 3 个过程,算法的复

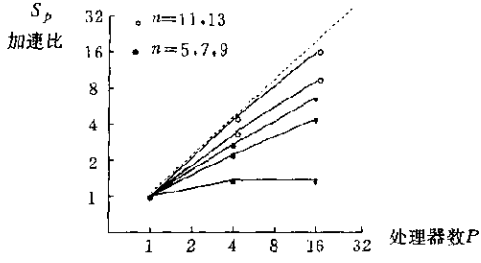

图6 加速比曲线

结果是否比串行结果好，该工作有待于进一步探索.

杂度主要在排序阶段；第3，我们的算法是一个并行线性四分树算法，并采用了模拟四分树遍历的算法技术，使串行的时间复杂度 $O(B\times\log B)$ 降为 $O((B\times\log B)/P)$.

虽然我们假设所有图象是二进制的，但是对彩色图象该算法同样有效. 以往的转换算法的时间复杂度是 $O(n\times B)$[5~6]，n 是树的深度，B 是循环代码长度. 而我们的串行算法时间复杂度为 $O(B\times\log B)$. 对于一般图象边界不会充满整个图象，$n\leq\log B$，因而时间复杂度 $O(B\times\log B)$ 一般优于 $O(n\times B)$. 目前最好的串行时间算法为 $O(n+B)$，但它有约束条件，而且尚不清楚它的并行

参考文献

1 Zahid Hussain. Digital image processing: ractical applications of parallel processing techniques. England: Ellis Horwood Limited, 1991

2 Same H. Applications of spatial data structures: computer graphics, image processing, and GIS. Reading, MA: Addison-Wesley, 1989

3 Frank Dehne, Andeew Rau-Chaplin. Hypercube algorithm for parallel processing of pointerbased quadtrees. Computer Vision and Image Understanding, 1995,52(1):1~10

4 Gargantini I. An effective way to represent quadtrees. Communications of the ACM, 1982,25:905~910

5 Samet H. Region representation: quadtrees from boundary codes. Communications of the ACM, 1980,23:163~170

6 Atkinson H H, Gargantini I, Walsh T S R. Filling by quadrants or octants. Computer Vision Graphics Image Process, 1986, 33,138~155

7 Mark D, Abel D. Linear quadtrees from vector representations of polygons. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1985,7:344~349

8 Chen Guo-liang. Design and analysis of parallel algorithms. Beijing: High Education Press, 1994

9 Chen Guo-liang. Parallel algorithms: sorting and selection. Hefei: University of Science and Technology of China Press, 1990

10 Anguh M M, Martin R R. A truncation method for computing walsh transforms with applications to image processing. CVGIP: Graphical Models and Image Processing: 1993,55(6):482~493

A Parallel Algorithm and Implementation of Boundary Conversion in Image Processing

YANG Bo CHEN Hu CHEN Guo-liang

(*Department of Computer Science University of Science and Technology of China Hefei 230027*)

Abstract This paper presents a parallel method of converting boundary to region quadtree. The method based on MIMD model has been implemented on DAWN1000 by comparing sequential results with the parallel results. The algorithm can be finished in $O((B\times\log B)/P)$, where B is the number of chaincodes, and P is the number of processors. The algorithm can be applied widely in image processing, computer graphics, and pattern recognition etc.

Key words Quadtree, Morton code, Jordan curve, chaincode, quick sort.

Class number TP391.41