1000-9825/2001/12(02)0233-08 ©2001 Journal of Software 软件学报 Vol.12, No.2

虫孔路由 Mesh 上的连通分量算法及其应用*

许胤龙， 万颖瑜， 顾晓东， 陈国良

(中国科学技术大学 计算机科学技术系,安徽 合肥 230027);

(国家高性能计算中心,安徽 合肥 230027)

E-mail: ylxu@ustc.edu.cn

摘要：用倍增技术在带有 Wormhole 路由技术的 $n\times n$ 二维网孔机器上提出了时间复杂度为 $O(\log^2 n)$ 的连通分量和传递闭包并行算法，并在此基础上提出了一个时间复杂度为 $O(\log^3 n)$ 的最小生成树并行算法. 这些都改进了 Store-and-Forward 路由技术下的时间复杂度下界 $O(n)$. 同其他运行在非总线连接分布式存储并行计算机上的算法相比，此连通分量和传递闭包算法的时间复杂度是最优的.

关键词：连通分量；图论算法；并行算法；虫孔路由；网孔机器

中图法分类号：TP393 **文献标识码**：A

设 $G=(V,E)$ 是无向图(文中记 $n=|V|$). 求图 G 的连通分量、传递闭包、最小生成树是图论中的基本问题，由于其广泛的应用，这些问题吸引了许多专家学者的注意，有着许多的研究成果[1~10]. 陈国良等人在文献[1]中总结了共享存储模型下的最新成果. 对于求图的连通分支和最小生成树，Shiloach[3]等人基于 CRCW(concurrent-read and concurrent-write)给出了时间复杂度为 $O(\log n)$、$O(m+n)$ 个处理器的算法. Johnson[4]等人在 1991 年基于 CREW(concurrent-read and exclusive-write)给出了一个时间复杂度为 $O(\log^{3/2} n)$、处理器个数为 $O(m+n)$ 的算法. 该算法第一次打破了在 CREW 上这一问题的时间复杂度界限 $O(\log^2 n)$. 在此基础上，Chong[5]等人基于 EREW(exclusive-read and exclusive-write)在 1993 年进一步提出了时间复杂度为 $O(\log n\log\log n)$、处理器个数为 $O(m+n)$ 的算法. 这至今仍是最好的结果. 图的并行算法在 20 世纪 80 年代曾是算法研究的一个热点. 对于图的连通分量、传递闭包和最小生成树，在各种拓扑结构的互连网络上也有许多研究成果[6~8]. 梁维发等人[6]和 Huang[7]分别在 $p(p\log p\leqslant n)$ 个处理器的超立方体上和 $p(p\leqslant n^2/\log^2 n)$ 个处理器的树网上提出了时间复杂度为 $O(n^2/p)$ 的并行算法；在 $n\times n$ 的二维网孔机器(Mesh)上，Nassimi 等人[8]提出的连通分量算法和 Maggs 等人[9]提出的最小生成树算法的运行时间为 $O(n)$. 就我们所知，目前在带有 Wormhole 路由器的并行机上还没有这方面的研究成果.

目前，Wormhole 路由技术已很成熟，许多二维网孔机器上都采用了 Wormhole 路由技术，如我国的曙光系列并行机等. 这促进了在带有 Wormhole 路由技术的二维 Mesh 上算法的研究，也取得了不少成果，但其中的绝大多数都是有关各类路由算法的，在这种模型上进行的应用算法和算法理论的研究却很少[11]. 而此类研究的理论和应用价值却是显然的. 文中用倍增技术

* 收稿日期：1999-08-04；修改日期：1999-11-12

基金项目：国家教育部博士点基金资助项目(9703825)

作者简介：许胤龙(1963－)，男，安徽庐江人，副教授，主要研究领域为复杂性理论，并行算法；万颖瑜(1976－)，男，江西南昌人，博士生，主要研究领域为复杂性理论，并行算法；顾晓东(1975－)，男，江苏无锡人，博士生，主要研究领域为复杂性理论，并行算法；陈国良(1938－)，男，安徽颍上人，教授，博士生导师，主要研究领域为并行与分布式算法，遗传算法.

(pointer-jumping)在带有 Wormhole 路由技术的 $n\times n$ 二维 Mesh 上提出了时间复杂度为$O(\log^2 n)$的连通分量和传递闭包并行算法,并在此基础上提出了一个时间复杂度为 $O(\log^3 n)$的最小生成树并行算法.这些都改进了 Store-and-Forward 路由技术下的时间复杂度下界 $O(n)$.就我们所知,在带有 Wormhole 路由技术的二维网孔机器上,这些都是第 1 个时间复杂度为对数的多项式级的算法.与其他运行在非总线连接分布式存储并行计算机上的算法相比,本文的连通分量和传递闭包算法的时间复杂度是最优的.

本文第 1 节介绍带有 Wormhole 路由技术的二维网孔机器,第 2 节介绍图的连通分量算法,第 3 节利用第 2 节的连通分量算法给出传递闭包和最小生成树算法,最后是总结.

1 带有 Wormhole 路由技术的二维网孔机器

一个 $n\times n$ 的二维网孔机器是将 n^2 个处理器排成 $n\times n$ 的方阵,然后在行、列方向相邻的两个处理器之间加一条通信链.Store-and-Forward 路由技术只允许相邻的两结点间进行通信,若两个待通信结点间的距离为 d,则数据在整个传送过程中被复制了 $d-1$ 份,其通信时间与 d 成正比.因为$n\times n$的二维网孔机器的直径为 $2n-1$,因而在它上面用 Store-and-Forward 路由技术实现的任何算法都不可能突破时间下界 $O(n)$.

在一个互连网络上用 Wormhole 路由技术事先在两个待通信结点间建立一条路径,然后由该条路径直接将数据发到目标结点.整条路径在发送过程中被两个结点独占,该路径上其余的结点在此时不能同任何其他结点进行通信.若忽略线长对通信的延迟,则我们可以假定任意两结点间传送单位长数据的延迟为单位时间.本文的算法建立在 Wormhole 路由技术的基础上,下面将不再特别加以说明.

下面,我们首先介绍后面要用到的二维 Mesh 上的行列播送算法和求最小值算法.

1.1 行列播送算法

设在 $n\times n$ 的二维网孔机器上处理器按行、列编号为 $P_{ij}(1\leqslant i,j\leqslant n)$,若要将处理器 P_{i1}中的数据 d 播送到第 i 行所有的处理器,则可先将 d 经过路径 $P_{i1}-P_{i2}-...-P_{i,\frac{n}{2}-1}$从 P_{i1}播送到 $P_{i,\frac{n}{2}+1}$,然后 P_{i1} 与 $P_{i,\frac{n}{2}+1}$ 同时分别递归地将 d 播送到处理器 $P_{i1},P_{i2},...,P_{i,\frac{n}{2}}$ 和处理器 $P_{i,\frac{n}{2}-1}$, $P_{i,\frac{n}{2}-2},...,P_{in}$. 整个播送过程需 log 步.

对于第 i 行的某一处理器 P_{ij},若要将其上的数据播送到第 i 行所有的处理器,可先将其传到 P_{i1}上.然后再按以上方式传送.列播送方法可与行播送相同.因而我们有下面的引理.

引理 1. 在 $n\times n$ 的 Wormhole 路由二维网孔机器上可在 $\log n+1$ 步内实现行、列播送.

1.2 求最小值算法

设在 $n\times n$ 的二维网孔机器上第 i 行的处理器 $P_{i1},P_{i2},...,P_{in}$中分别含有数据 $d_{i1},d_{i2},...,d_{in}$,我们要求的是 $d_{i1},d_{i2},...,d_{in}$中的最小值.为此,列编号为奇数的处理器 $P_{i,2j-1}$将其中的数据 $d_{i,2j+1}$送到列编号为偶数的处理器 $P_{i,2j+2}$,$P_{i,2j+2}$作比较 $\min\{d_{i,2j+1},d_{i,2j+2}\}(j=0,1,...,n/2-1)$.然后处理器 $P_{i,4j+2}$将 $\min\{d_{i,4j+1},d_{i,4j+2}\}$送到处理器 $P_{i,4j+4}$,$P_{i,4j+4}$比较 $\min\{d_{i,4j+1},d_{i,4j+2}\}$和 $\min\{d_{i,4j+3},d_{i,4j+4}\}$的大小$(j=0,1,...,n/4-1)$,...,如此递归地执行 $\log n$ 次即可求出 d_{i1}, $d_{i2},...,d_{in}$中的最小值.对于列,情况类似.因此有下面的引理.

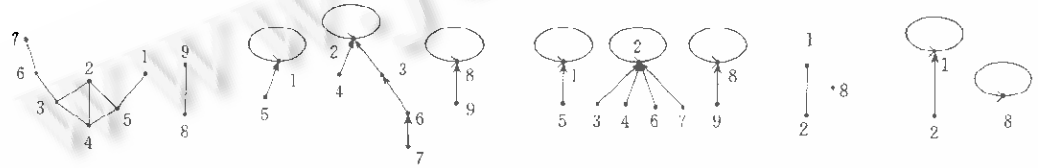
引理 2. 设在 $n\times n$ 的 Wormhole 路由二维网孔机器上某行或某列的每一个处理器中含有一个数据,则可在 $\log n$ 步求出其最小值.

2 连通分量算法

设无向图 $G=(V,E)$ 的邻接矩阵为 $A=(a_{ij})_{n\times n}$,a_{ij} 存于 $n\times n$ 的二维Mesh的处理器 P_{ij} 中,我们用倍增技术在二维Mesh上求 G 的连通分量.文献[2]中介绍了用类似的思想在PRAM CRCW上实现了连通分量算法.下面首先简单介绍用倍增技术求连通分量的基本思想,然后再详细介绍在 $n\times n$ 的二维Mesh上的实现.

2.1 算法思想

在下面的讨论中我们假设 $a_{ii}=1(i=1,2,\ldots,n$,即假定每个顶点是其自身的邻顶点),在 V 上定义函数 C:$C(v)=\min\{u\mid a_{uv}=1\}$,即 $C(v)$ 是 v 的所有邻顶点(包括 v 自身)中的最小者.以 V 为顶点集,对 V 中任一顶点 v,从 v 向 $C(v)$ 引一条有向边,这就构成了一个有向图 G',如图1(b)所示(图1(a)为图 G).因为 G' 中每个顶点的出度都为1,所以 G' 的每个连通片都是一棵根部带环的内向树(以下简称 G' 为有向森林,而称 G' 的每个连通片为树),而且易知有以下引理.



(a) (b) (c) (d) (e)

Fig.1
图1

引理3. 设 G' 中所有的有向树为 $T_1,T_2,\ldots,T_s$,相应的树根和顶点集分别为 $r_1,r_2,\ldots,r_s$ 和 $V_1,V_2,\ldots,V_s$,则 $r_i(i=1,2,\ldots,s)$ 是 V_i 中的最小顶点且每个 $V_i(i=1,2,\ldots,s)$ 中所有的顶点都在 G 的同一个连通片中.

证明:略.文献[2]中有与此类似的结论.

在 G' 中用倍增技术(pointer-jumping)使 G' 中每个顶点都指向该顶点所在有向树的根,即对 G' 中每个顶点 v,循环令 $C(v):=C(C(v))$ 最终使 $C(v)=C(C(v))=$ 顶点 v 所在树的树根 r_i,如图1(c)所示.这样,通过 $C(v)$ 将 v 标记为 G' 中 v 所在树的最小顶点,这可以在 $O(\log n)$ 时间内完成[2].

由引理3可知,每个 $V_i(i=1,2,\ldots,s)$ 中的所有顶点都在 G 的同一个连通片中,但不同 V_i 中的顶点也有可能在 G 的同一个连通片中,如图1(b)中的顶点1和顶点2.因此,还需要将处在 G 的同一个连通片却在不同 V_i 中的顶点标识成同一个最小顶点.为此,我们将每个 $V_i(i=1,2,\ldots,s)$ 看做一个顶点,称作超顶点,用 V_i 的根 r_i 来表示.若存在 $v\in V_i$ 和 $w\in V_j$,使得 v 和 w 在 G 中相邻,则在超顶点 r_i 和 r_j 间连一条边,这样就构成一个无向图,下文中称之为超图,记为 H,如图1(d)所示.容易验证,两顶点 v,w 在 G 的同一个连通片中等价于 v,w 所在有向树的根所对应的超顶点在超图 H 的同一连通片中.为了求 G 的连通分量,我们还需做如下工作:

(1) 求超图 H.

(2) 用无向图 G 导出有向森林 G' 的方法,由超图 H 导出有向森林 H'.如图1(e)所示.

(3) 在 H' 中用倍增技术使每个超顶点指向该超顶点所在树(在 H' 中)的根结点,成为该超顶点的新标记.

(4) 将超顶点的新标记传至该超顶点在 G' 中相应的树的每个顶点.

循环执行(1)～(4)步直到 H 中所有的顶点都是孤立顶点为止.最终 G 中每个顶点的标记就是该顶点所在连通片的最小顶点(关于正确性,可参见文献[2]).在每次循环中,H 中的每个非孤立顶点至少与另外一个非孤立顶点合并成一个超顶点,所以(1)～(4)步至多执行 $O(\log n)$次.

2.2 在二维 Mesh 上的实现

设无向图 $G=(V,E)$的邻接矩阵为 $A=(a_{ij})_{n\times n}$,其中 $V=\{1,2,...,n\}$,a_{ij}存于 $n\times n$ 二维网孔机器的处理器 P_{ij}中.本节我们详细讨论第 2.1 节中的算法思想在二维 Mesh 上的具体实现.在下面的算法中,我们给出了一种在二维 Mesh 上实现的数据播送方法,使得每次倍增可以在 $O(1)$时间内完成,从而使整个算法的时间复杂度为 $O(\log^2 n)$.算法的形式描述如下:

算法. Connected-Component

输入:无向图 G 的邻接矩阵 $A=(a_{ij})_{n\times n}$,a_{ij}存于 P_{ij}中.

输出:顶点标记 c_{ii},表示 G 中顶点 i 所在连通片的最小顶点,$i=1,2,...,n$.

```
Begin
(1) for i,j=1 to n pardo                          /* 初始化 */
       if i=j then c_ij:=1 else c_ij:=a_ij        /* 将每个顶点定义为与其自身相邻 */
    endfor
(2) for i=1 to n pardo                            /* 建立有向森林 */
       c_ii:=min{j|c_ij=1,j=1,2,...,n}            /* c_ii是顶点 i 的最小邻顶点 */
    endfor
(3) 用倍增技术将每个顶点指向树根.
    (3.1) for i=1 to n pardo
             将 c_ii播送到第 i 行所有的处理器      /* 用于下列循环中解决读冲突 */
          endfor                                  /* 以保证每次倍增能在 O(1)时间完成 */
    (3.2) 下一步循环执行 O(logn)次.
          for i=1 to n pardo
             (3.2.1) 处理器 P_ii沿纵向通信链向处理器 P_{c_ii i}发送读请求.
             (3.2.2) 处理器 P_{c_ii i}沿纵向通信链向处理器 P_ii发送数据 c_{c_ii c_ii}.
             (3.2.3) 处理器 P_ii令 c_ii:=c_{c_ii c_ii}.   /* P_ii读取 P_{c_ii c_ii}的内容,实现一次倍增 */
          endfor
下面第(4)～(7)步循环 O(logn)次.                    /* 每次先建立一个超图,再用倍增技术 */
(4) for i=1 to n pardo
       r_i:=c_ii                                  /* 中间变量,用以标记上一次循环后的超顶点和非超顶点 */
    endfor                                        /* i=r_i 的顶点为超顶点,否则为非超顶点 */
(5) for i=1 to n pardo                            /* 建立与超图 H 对应的有向森林 H' */
    (5.1) 将 c_ii播送到第 i 行所有的处理器.
    (5.2) 将 c_ii播送到第 i 列所有的处理器.
    (5.3) 在第 i 行求 d_i:=min{c_jj|a_ij=1,j=1,2,...,n},并置于处理器 P_ii.
                                                  /* 在每个顶点的所有邻顶点的标号中求出最小者 */
    (5.4) P_ii沿横向通信链将 d_i 送到 P_{i r_i}.
                 /* 将同一棵树中所有顶点的邻顶点标号最小者送到超顶点(树根)r_i 所在的列 */
    (5.5) 在所有 i=r_i 的列求 c_ii:=min{d_j|,j=1,2,...,n}.
                        /* i=r_i 的顶点为超顶点(树根),在超顶点间连有向边建立 H' */
```

endfor

(6) 对有向森林 H' 用倍增技术，即对所有 $i=r_i$ 的顶点（超顶点）（在二维 Mesh 上体现为相应的行列操作，其余的行列不操作）用与第 3 步相同的程序.

(7) 将每个超顶点（树根）的新顶点标号 c_{ii} 播送到相应树中的所有顶点.

(7.1) 在所有 $i=r_i$（超顶点）的行将 c_{ii}（i 的新顶点标号）播送到该行的每一个处理器.

/* 用以解决(7.2)步中的读冲突 */

(7.2) 对所有 $i\neq r_i$（非超顶点）的处理器 P_{ii} 沿纵向通信链向处理器 $P_{r_i i}$ 发读请求.

(7.3) 接受到读请求的处理器 $P_{r_i i}$ 向处理器 P_{ii} 发 $c_{r_i r_i}$，P_{ii} 令 $c_{ii}:=c_{r_i r_i}$.

End

下面对算法 Connected-Component 给出一些必要的说明，并给出复杂度分析.

算法的第(2)步在图 G 中求出了顶点 $i(i=1,2,...,n)$ 的最小邻顶点 c_{ii} 并存于处理器 P_{ii}，建立了顶点 i 到 c_{ii} 的有向边. 由引理 2 可知，第(2)步所需时间为 $O(\log n)$. 算法的第(3.1)步将 $c_{ii}(i=1,2,...,n)$ 播送到第 i 行的每一个处理器，形成 n 个备份以解决第(3.2)步中的读冲突，由引理 1 可知，所需时间为 $O(\log n)$. 算法的第(3.2)步每执行一次实现一次倍增(pointer-jumping)，设顶点 i 当前的标记为顶点 c_{ii}，则一次倍增后顶点 i 的标记是顶点 c_{ii} 当前的标记 $c_{c_{ii}c_{ii}}$，在二维 Mesh 中即需要将当前处于处理器 $P_{c_{ii}c_{ii}}$ 中的数据 $c_{c_{ii}c_{ii}}$ 传到处理器 P_{ii} 中. 由于可能存在不同的顶点，如 u 和 v，使得它们的当前顶点标记 c_{uu} 和 c_{vv} 相同，这样，在倍增时处理器 P_{uu} 和 P_{vv} 要同时读处理器 $P_{c_{uu}c_{uu}}$（$=P_{c_{vv}c_{vv}}$）中的数据 $c_{c_{uu}c_{uu}}$（$=c_{c_{vv}c_{vv}}$），为了避免冲突在第(3.1)步形成了备份，这样，P_{uu} 和 P_{vv} 就可以沿不同的纵向通信链分别到处理器 $P_{c_{uu}u}$ 和 $P_{c_{vv}v}$ 中取 $c_{c_{uu}c_{uu}}$（$=c_{c_{vv}c_{vv}}$），因而就不存在冲突，使得一次倍增可以在 $O(1)$ 时间内完成. 第(3.2)步所需时间为 $O(\log n)$.

算法的第(5)步建立了与超图 H 相应的有向森林 H'. 由引理 1 可知，第(5.1)步和第(5.2)步所需时间均为 $O(\log n)$. 在执行完第(5.1)步和第(5.2)步后，处理器 $P_{ij}(i,j=1,2,...,n)$ 中含有顶点 i,j 的当前顶点标号 c_{ii} 和 c_{jj}（分别为顶点 i,j 所在树的树根（超顶点））. 第(5.3)步在顶点 $i(i=1,2,...,n)$ 所有邻顶点的标号（邻顶点所在树的树根，即超顶点）中找出最小的标号 d_i（超顶点）. 第(5.4)步处理器 P_{ii} 将顶点 i 的最小邻顶点标号 d_i 送到处理器 P_{ir_i}，这样，对于每个超顶点（树根）r_i 来说，第 r_i 列的处理器中包含了 r_i 所在树中所有顶点的最小邻顶点标号（邻顶点所在树的树根，即超顶点）. 而第(5.5)步则求出了每个超顶点在超图 H 中的最小邻顶点，从而建立了有向森林 H'. 第(5.4)步所需时间为 $O(1)$，由引理 2 可知，第(5.3)步和第(5.5)步均需 $O(\log n)$ 步. 所以，第(5)步每执行一次耗时 $O(\log n)$.

算法的第(6)步仅对超顶点所在的行和列进行操作，而对其余的行列不操作. 算法同第(3)步.

算法的第(7)步将每个超顶点的新标号送到超顶点（树根）所在树中所有的顶点. 第(7.1)步首先将超顶点的新顶点标号送到所在行所有的处理器中，形成备份以解决第(7.2)步和第(7.3)步中的读写冲突. 设非超顶点 i 的当前顶点标号为 r_i（超顶点，i 所在树的树根），第(7.2)步和第(7.3)步实现了将顶点 r_i 的新顶点标号 $c_{r_i r_i}$ 从处理器 $P_{r_i i}$ 向处理器 P_{ii} 的传送. 将其作为 i 的新顶点标号. 整个第(7)步所需时间为 $O(\log n)$.

由于第(4)～(7)步循环执行了 $O(\log n)$ 次，所以，整个算法所需时间为 $O(\log^2 n)$. 综上所述，我们得到下面的定理.

定理 1. 算法 Connected-Component 结束时，处理器 $P_{ii}(i=1,2,...,n)$ 中含有的数据 c_{ii} 即为顶点 i 所在连通片中的最小顶点. 整个算法的执行时间为 $O(\log^2 n)$.

3 传递闭包和最小生成树算法

3.1 传递闭包算法

设 G 为无向图,其邻接矩阵为 $A=(a_{ij})_{n\times n}$,其中 $a_{ij}(i,j=1,2,...,n)$存于二维 Mesh 的处理器 P_{ij}中,要在二维 Mesh 上求出 G 的传递闭包 G^*,G^* 以其邻接矩阵 $A^*=(a_{ij}^*)_{n\times n}$来表示,计算结果是将 a_{ij}^*存于处理器 P_{ij}中.

设已求出 G 的连通分量,顶点 $i(i=1,2,...,n)$所在的连通分量的最小顶点为 c_{ii},c_{ii}存于处理器 P_{ii}中,则我们可用下述简单的方法求 G^*:

(1) 在所有的行并行地做:将 $c_{ii}(i=1,2,...,n)$播送到第 i 行的每一个处理器.

(2) 在所有的列并行地做:将 $c_{jj}(j=1,2,...,n)$播送到第 j 列的每一个处理器.

(3) 所有的处理器 $P_{ij}(i,j=1,2,...,n)$并行地做:若 $c_{ii}=c_{jj}$,则令 $a_{ij}^*=1$;否则,令 $a_{ij}^*=0$.

在执行了算法的第(1)和第(2)步后,若处理器 P_{ij}中含有顶点 i 和 j 的最小连通分量标号两者相同,则说明顶点 i 和 j 在 G 的同一个连通分量中,从而在 G^* 中 i 和 j 相邻,故 $a_{ij}^*=1$;否则,i 和 j 在 G^* 中不相邻,故 $a_{ij}^*=0$. 由引理 1 可知,上述方法的执行时间为 $O(\log n)$.

若仅有 G 的邻接矩阵存于二维 Mesh 中,而并不知道 G 的连通分量,则可先用第 2 节的算法 Connected-Component 求出 G 的连通分量,然后再用上述方法. 因而有下面的定理.

定理 2. (1) 若已知无向图 G 的连通分量,顶点 $i(i=1,2,...,n)$的连通分量标号 c_{ii}存于二维 Mesh 的处理器 P_{ii}中,则可在二维 Mesh 上用 $O(\log n)$时间求出 G 的传递闭包.

(2) 设无向图 G 的邻接矩阵为 $A=(a_{ij})_{n\times n}$,其中 $c_{ij}(i,j=1,2,...,n)$存于二维 Mesh 的处理器 P_{ij}中,则可在二维 Mesh 上用 $O(\log^2 n)$时间求出 G 的传递闭包.

3.2 最小生成树算法

设 G 是带有边权 W 的无向图,其中边(i,j)的权 $W(i,j)$(若顶点 i 和 j 在 G 中不相邻,则记 $W(i,j)=\infty$)存于 $n\times n$ 的二维 Mesh 的处理器 P_{ij}中. 要求 G 的最小生成树 T,我们用第 3.1 节的传递闭包算法将 Sollin 算法[10]在二维 Mesh 上并行地实现.

Sollin 算法循环执行不多于 $O(\log n)$次. 算法开始时将图的每个顶点看作一棵树,这样就有了一个由 n 个孤立顶点构成的森林. 在算法的每次循环中:(1) 对每棵树求出一端在该树中,而另一端不在该树中的最短边;(2) 将(1)中求出的最短边加进当前森林,对当前森林中的树进行合并,直到森林中仅有一棵树为止. 由于每次循环时至少将两棵树合成一棵,所以,Sollin 算法至多循环执行$O(\log n)$次.

图 G 的最小生成树 T 在二维 Mesh 上以 T 的边权矩阵来表示:若边(i,j)是 T 的边,则处理器 P_{ij}含有 T 的边权 $T(i,j)=W(i,j)$,否则有 $T(i,j)=0$. 在二维 Mesh 上实现的最小生成树算法的基本思想如下:

(1) 将 G 的最小生成树 T 初始化为 n 个孤立的顶点,并将每个顶点的连通分量标记为其自身. 即当 $i\neq j$ 时,P_{ij}令 $T(i,j)=0$;当 $i=j$ 时,P_{ii}令 $T(i,i)=i$.

(2) 下列各步循环 $O(\log n)$次.

(2.1) 在 Mesh 所有的行并行地做:在 G 中求出顶点 i 连出的最短边(i,j_i).

(2.2) 设顶点 i 的当前连通分量标号为 c_{ii},将顶点 i 连出的最短边(i,j_i)的权 $W(i,j_i)$沿横向通信链从处理器 P_{ij_i}传至处理器 $P_{ic_{ii}}$. /* 将所有标号为 c_{ii}的树中的顶点所连出的最短边权全部传到第 c_{ii}列 */

(2.3) 在所有 $i=c_{ii}$ 的列求出每棵树连出的最短边,并置于处理器 $P_{ii}(=P_{c_{ii}c_{ii}})$ 中.

(2.4) 将在第(2.3)步求出的最短边加进森林中,即若 (i,j) 是第(2.3)步求出的最短边(由第(2.3)步可知,(i,j) 存于处理器 $P_{c_{ii}c_{ii}}$ 中),则先从处理器 $P_{c_{ii}c_{ii}}$ 沿纵向通信链向处理器 $P_{ic_{ii}}$ 发一个信号,然后再从处理器 $P_{ic_{ii}}$ 沿横向通信链将信号转发给处理器 P_{ij},每个接受到该类信号的处理器置 $T(i,j)=W(i,j)$.

(2.5) 用第 3.1 节的传递闭包算法求出当前森林的传递闭包.这里是边权矩阵而不是邻接矩阵,因而在用算法 Connected-Component 时需将顶点相邻的定义修改为 $T(i,j)\neq 0$.

(2.6) 将 G 中属于森林的传递闭包的边权记为∞,即若 (i,j) 是传递闭包的边,则处理器 P_{ij} 置 $W(i,j)=\infty$. /* 将同一棵树中的边删掉,以保证第(2.1)步求出的最短边一定不在同一棵树内 */

(3) 并行执行:$T(i,i)=0$. /* (i,i) 不是 T 的边,而在第(2.5)步中却使 $T(i,i)\neq 0$ */

上述算法的正确性不难证明(略).由定理 2 可知,算法的第(2.5)步所需时间为 $O(\log^2 n)$,而由引理 2 可知,算法的第(2.1)步和第(2.3)步所需时间为 $O(\log n)$,其余各步的执行时间均为 $O(1)$.因而有下面的定理.

定理 3. 设 G 是带有边权 W 的无向图,其中边 (i,j) 的权 $W(i,j)$(若顶点 i 和 j 在 G 中不相邻,则记 $W(i,j)=\infty$)存于 $n\times n$ 的二维 Mesh 的处理器 P_{ij} 中,则可在 $O(\log^3 n)$ 时间求出 G 的最小生成树.

4 总 结

文中用倍增技术(pointer-jumping)在带有 Wormhole 路由技术的 $n\times n$ 的二维 Mesh 上提出了时间复杂度为 $O(\log^2 n)$ 的连通分量和传递闭包并行算法,并提出了一个时间复杂度为 $O(\log^3 n)$ 的最小生成树并行算法.这些都改进了 Store-and-Forward 路由技术下的时间复杂度下界.与其他运行在非总线连接分布式存储的并行计算机上的算法相比,本文所给出的连通分量和传递闭包算法的时间复杂度是最优的.目前,Wormhole 路由技术已被广泛应用于各类并行机,它改进了网孔连接的并行机的通信性能,因而我们认为在这方面有着良好的研究前景.

References:

[1] Chen, Guo-liang, Liang, Wei-fa, Shen, Hong. Research advances in parallel graph algorithms. Computer Research and Development, 1995,32(9):1~16 (in Chinese).

[2] Chen, Guo-liang. Design and Analysis of Parallel Algorithms. Beijing: Higher Education Press, 1994 (in Chinese).

[3] Shiloach, Y., Vishkin, U. An $O(\log n)$ parallel connectivity algorithms. Journal of Algorithms, 1982,3(1):57~67.

[4] Johnson, D.B., Metaxas, P. Connected components in $O(\log^{3/2}|V|)$ parallel time for the CREW PRAM. In: Proceedings of the FOCS'91. Los Angeles: IEEE Computer Society Press, 1991. 688~697.

[5] Chong, K.W., Lam, T.W. Connected components in $O(\log n\log\log n)$ time on CREW PRAM. In: Proceedings of the 4th Annual ACM SIAM Symposium on Discrete Algorithms. Austin, Texas: ACM Press, 1993. 11~20.

[6] Liang, Wei-fa, Chen, Guo-liang. Optimal graph algorithms on multiprocessor. Chinese Journal of Computers, 1991,14(9):641~650 (in Chinese).

[7] Huang, D.M. Solving some graph problems with optimal or near-optimal speedup on mesh of trees network. In: Proceedings of the 26th Annual IEEE FOCS. Taipei: IEEE Computer Society Press, 1985. 232~240.

[8] Nassimi, D., Sahni, S. Finding connected components on mesh-connected parallel computers. SIAM Journal on Computing, 1980,9(4):744~757.

[9] Maggs, B.M., Plotkin, S.A. Minimum-Cost spanning tree as a path finding problem. Information Processing Letters, 1988,26(6):291~293.

[10] Sollin, M. An algorithm attributed to sollin. In: Goodman, S.E., Hedetniemi, S.T., eds. Introduction to the Design and

Analysis of Algorithms. New York: McGraw-Hill, Inc., 1977.

[11] Xu, Yin-long, Wang, Xun. Parallel *K*-selection on wormhole routed 2D Mesh. Chinese Journal of Computers, 1999,22(12):1309~1313 (in Chinese).

附中文参考文献:

[1] 陈国良,梁维发,沈鸿.并行图论算法研究进展.计算机研究与发展,1995,32(9):1~16.

[2] 陈国良.并行算法的设计与分析.北京:高等教育出版社,1994.

[6] 梁维发,陈国良.多处理器上图的最优算法,计算机学报,1991,14(9):641~650.

[11] 许胤龙,王洵.基于Wormhole路由的二维Mesh上的并行*K*-选择.计算机学报,1999,22(12):1309~1313.

Connected Component Algorithm on Wormhole Routed Mesh and Its Applications*

XU Yin-long, WAN Ying-yu, GU Xiao-dong, CHEN Guo-liang

(*Department of Computer Science and Technology, University of Science and Technology of China, Hefei* 230027, *China*);
(*National High Performance Computing Center at Hefei, Hefei* 230027, *China*)
E-mail: ylxu@ustc.edu.cn

Abstract: A connected component and transitive closure parallel algorithm using pointer jumping technique is presented in this paper, which runs on $n\times n$ wormhole routed 2D mesh in time $O(\log^2 n)$. A minimum spanning tree (MST) parallel algorithm running on the same model in time $O(\log^3 n)$ is also presented. These improve the lower time bound $O(n)$ on $n\times n$ store-and-forward routed 2D mesh. Compared with other known algorithms running on various non-bus-connected parallel machines with distributed memory, the time complexity of the connected component and transitive closure parallel algorithm is optimal.

Key words: connected component; graph algorithm; parallel algorithm; wormhole routing; mesh

* Received August 4, 1999; accepted November 12, 1999
Supported by the National Research Foundation for the Doctoral Program of Higher Education of China under Grant No. 9703825