文章编号:1001 - 9081(2005)09 - 2111 - 03

基于 Hypercube 的最长导出路算法

邱成功,马英红,任晓慧
(山东师范大学 信息管理学院,,山东 济南 250014)
(chgqiu@163.com)

摘 要:超立方体(Hypercube)网络是多处理机系统中常见的一种互连网络。文中提出 Hypercube 中最长导出路的概念,然后给出一种能改善多处理机系统中传输容错性的最长导出路算法,最后给出该算法的C++实现。

关键词:超立方体;容错;多处理机系统;C++
中图分类号: TP393.01 **文献标识码**:A

Algorithm for the longest induced path in Hypercube

QIU Cheng-gong, MA Ying-hong, REN Xiao-hui
(*College of Information Management, Shandong Normal University, Ji'nan Shandong* 250014, *China*)

Abstract: Hypercube network is a kind of common interconnection network. The definition of the longest induced path (the LIP) in hypercube was proposed with an algorithm for the LIP which could improve the performance of the fault-tolerance in multiprocessors. In the end, the realization of the algorithm using the language C++ was given.

Key words: Hypercube; fault-tolerance; multiprocessors; C++

0 引言

超立方体网络(Hypercube)是多处理机系统中常见的一种互连网络,由 Squir 和 Palais 于 1963 年最早提出。这种拓扑结构由于具有直径小、可扩展性强、结构对称、网络寻路算法简单等特点,且许多互连网络,如环形网络、树形网络以及 meshes 等等都可以在超立方体网络中很容易且高效率地得以实现,因而成为最重要和最具吸引力的网络模型之一,如 Intel iPSC/1、iPSC/2 和 nCUBE 机等并行机都采用了超立方体结构;而且,这种结构在实际的并行计算中也得到广泛应用。

但是,当 Hypercube 中的结点数目逐渐增多时,其传输过程中的容错性(fault-tolerance)能便成为一个非常关键的问题。为了提高并行系统的容错性,改善并行系统传输性能,我们提出了最长导出路的概念。

1 最长导出路

为了分析方便,在介绍最长导出路之前,我们先介绍以下基本概念和基础知识。

1.1 Hypercube 的定义及性质

定义 1 一个 n 维超立方体是一个无向图。它拥有 2^n 个结点(node),每一个结点位址可以用一个 n 位元的二进位数来表示,因此在结点上的标示范围从 0 到 $2^n - 1$。如果某两个结点的位址只有一个位元不同(即两个结点相邻),就会有一条边来连接这两个结点。

性质 1 n 维超立方体是一个规则图,每一各结点的度数(degree)均为 n。

性质 2 n 维超立方体具有递回性质,一个 n 维超立方体可以由 2 个 $n-1$ 维超立方体依照[定义 1]的方式增加 $2^n - 1$ 条边连接相对应的结点来组合而成。

例如,3 维超立方体(图 1 为我们规定的维数顺序,图 2 为该顺序下的 Hypercube 结点编号),共有 $2^3 = 8$ 个结点。每一个结点用 3 位元表示,标示范围从 0 到 7,分别为 000,001,011,010,100,101,111,110(为了后面算法和程序说明的方便,我们规定,编号从右向左依次为第 1 维、第 2 维、第 3 维)。因为每一个结点编号对应三个维数,故各个结点的度数亦为 3。

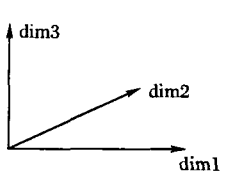

图 1 维数顺序图

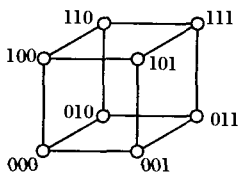

图 2 3 维 Hypercube 的结点及编号图

而该超立方体可以看作由 2 个 2 维超立方体(即正方形)按照[定义 1]的方式增加 4(即 $2^{(3-1)}$)条边连接对应的结点组合而成,示意图如图 3 所示。

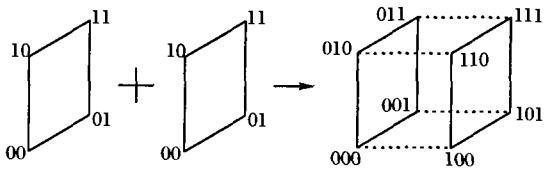

图 3 两个 2 维立方体构成 1 个 3 维超立方体图

1.2 Hypercube 中的最长导出路

定义 2 导出路为 Hypercube 中结点和结点相连的一条路径,在这条路径上,除左端结点只有右相邻结点、右端结点只有左相邻结点外,路径上的其他结点均有左右相邻结点;并且,导出路上每增加一个结点,该结点必须与其左相邻结点前面的所有结点均不相邻。

定义 3 最长导出路(LIP, the Longest Induced Path)即为 hypercube 中结点数最多(或者边数最多)的导出路。

收稿日期:2005 - 03 - 12;**修订日期**:2005 - 05 - 24 **基金项目**:国家自然科学基金资助项目(NSFC10471078)

作者简介:邱成功(1980 -),男,山东泰安人,硕士研究生,主要研究方向:计算机网络、路由算法; 马英红(1971 -),女,山东济南人,博士,副教授,主要研究方向:组合优化; 任晓慧(1978 -),女,山东聊城人,硕士研究生,主要研究方向:并行系统.

例如,在图 2 的 3 维 hypercube 中,000 -> 001 -> 011 -> 111 和 000 -> 100 -> 110 均为导出路,而 000 -> 001 -> 011 -> 111 -> 110 和 000 -> 100 -> 110 -> 111 -> 011 均为最长导出路(LIP)。

性质 3 在 n 维($n > 1$) hypercube 中,LIP 不是唯一的(最长导出路不只一条)。

1.3 LIP 的优点

在处理机间有效地传递信息是多处理机系统中关系到其性能的一个重要因素。由于多处理机系统处理机数目的不断增加,系统中出现处理机故障或处理机间的链路故障的可能性也随之增加,因此容错策略的设计对多处理机系统也越发显得重要。

最长导出路首先是由 hypercube 中的结点互连构成的,本身是一条通路,只要这条路径上的处理机没有故障,hypercube 中的其他处理机结点就都可以顺利正常地接收到数据,即使其他处理机结点有多个坏结点(因为 hypercube 中的任一结点要么是 LIP 上的结点,要么是和 LIP 上的结点相邻的,见下文[性质 4]);其次,由[性质 3],由于 LIP 不是唯一的,故如果一条 LIP 有故障的话,我们可以选择另外一条 LIP 来传输数据,因此可以大大提高多处理机系统的容错性能,从而提升系统应付可能的故障并保障在最恶劣情况下能保持其基本效能。

性质 4 在 hypercube 中,任一结点要么是 LIP 上的结点,要么与 LIP 上的结点相邻。

证明:即证对 $\forall v \in \{Q_n - V(LIP)\}$,都存在 $u \in V(LIP)$,使得 $uv \in E(Q_n)$。

这里,我们用 Q_n 来表示 hypercube 中的结点集合,用 $V(LIP)$ 来表示 LIP 的结点集合,用 $E(Q_n)$ 来表示 hypercube 的边集合,用图 4 来表示 LIP(共有 m 个结点):

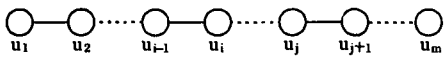

图 4 共有 m 个结点的 LIP 图

用反证法。

设有一结点 $v' \in \{Q_n - V(LIP)\}$,使得对 $\forall u \in V(LIP)$,均有 $uv' \notin E(Q_n)$。

v' 一定有它自己的去心邻域且大小为 n,不妨设为 $S_{v'}$ 且 $|S_{v'}| = n$,则 $S_{v'}$ 一定与某个 S_k 相交($k = 1,2,\cdots,m$)。假设 $S_{v'}$ 与 S_i 相交,公共元素为 w,由于 u_i 与 u_j 相邻时,$S_i \cap S_j = \varnothing$,故 $S_{v'} \cap S_j = \varnothing$,则把 w 加入原 LIP 中(即原来的 $u_i \rightarrow u_j$ 变为 $u_i \rightarrow w \rightarrow u_j$),新路径就比原来的路径长,故与题设相矛盾。

命题得证。

2 求 LIP 的算法

2.1 算法思想

该算法采用试探法,从设定的一个初始结点出发,按照 1.1 中图 1 的维数顺序,对 hypercube 中的结点逐个进行试探,满足条件(定义 2)就放入堆栈,然后继续试探,直到找不到满足条件的结点为止(即达到最长)。

2.2 算法描述

输入:Q_n(即 n 维 hypercube,包括结点、边及编号)

输出:LIP(即最长导出路)

算法:

① 给定起始点 S_0、堆栈 S = NULL,其深度 $dep = 0$,LIP 的长度标志 $len = 0$,并将 S_0 标记为当前点(当前点用 v 表示)。

② 找出 v 的邻域 D_v(即与 v 相邻的结点集合)。

③ 从 D_v 中,找出与堆栈 S 中所有点既不相同也不相邻的点,任取一个记为 v';若无这样的点存在,转 ⑤。

④ 将 v 入栈,并令 $v = v'$,转 ②。

⑤ 若 S 为空,转 ⑦,否则,若 $dep > len$,令 $dep -> len$,并令当前路径为 LIP。

⑥ 回退,即出栈 -> v,转 ②。

⑦ 输出 LIP(其结点数为 $len + 1$),结束。

分析:该算法采用试探的方式,从一个给定的起始点出发,通过分析其邻域,逐个试探当前点的邻结点是否满足要求,直到没有满足条件的可以加到该导出路上的点为止,即找到一条 LIP,算法思路清晰,结构严谨;但是,由于采用堆栈的结构,而且有些边进行了重复搜索,故复杂性较高(主要是时间复杂性)。

3 求 LIP 的C++实现

以下便是求 LIP 的C++程序,限于篇幅,只写出主程序,对其中用到的结构和变量稍作解释,以供参考。

```
void main()
{
  cout <<"Please input the dimension of the hypercube(3<= n<=6) : ";
  cin > > n;
  do
  {
    dep++; d = -1; b = false;
    do{
        d++;
    //依次试探每个方向,按第一维、第二维、第三维的维数顺序
        if( CanPass())                          //下一结点可以通过
        {
          stack[ dep]. x1 = x;
          stack[ dep]. d1 = d;                  //当前结点入栈
          x = x^dim[ d];
                  //移到下一结点,即下一结点变为当前结点
          if( IsFinished())                     //该条路径走完
          {
            k++;                                //计数器增 1
            ShowPath();                         //输出解
            Back();                             //回退,出栈
          }
          else
            b = true;
        }
        else                                    // 下一点通不过
        {
          if( d > = n) Back();          //如果维数超限,回退,出栈
          else b = false;
        }
    }while( b == false);
  }while( dep > = 1);
  cout < <"共有" < < k + 1 < <"组解" < < endl;
}
```

其中,stack 的定义为 struct stack0 {int x1; int d1;}; stack0 stack[100];为一结构体,存储 LIP 路径上的结点(最后一个结点除外,所以最后的 k 记录的是 LIP 的边数,本程序输

出结点数,故 $k+1$),$x1$ 为结点的编号,$d1$ 为找到该点时所走的维数;x, d 为当前结点编号及维数;n 为 hypercube 的维数,由用户输入;$dim[\,]$ 为一数组,用于控制变换维数的;dep 为堆栈指示器;函数 *bool CanPass( );bool NotIn(int value); bool NotNeighbour(int value); bool IsFinished( ); void Back( ); void ShowPath( );* 均为用户自定义的函数,分别用来判断当前结点是否可以通过、当前点在不在堆栈中、当前点和堆栈中的点是否相邻、当前路径是否走完、回退函数以及输出结果函数,限于篇幅,在此不做详细介绍。试验结果数据如表 1 所示。

表 1　由程序所得出的数据

维数/ n	结点数	边数	LIP 的条数
2	3	2	2
3	5	4	6
4	8	7	12
5	14	13	740
6	27	26	720

试验结果表明,随着 n 的增大,hypercube 容错的结点数越来越多(例如,$n=3$ 时,容错结点数为 2^3 - 5 = 3 = 2^(3 - 1) - 1;n = 4 时,容错结点数为 2^4 - 8 = 8 = 2^(4 - 1);而当 n = 6 时,容错结点数变为 2^6 - 27 = 37 = 2^(6 - 1) + 5。

4　结语

多处理机系统中的容错性一直是影响其性能的一个重要因素;文章在分析 hypercube 结构特点的基础上,提出最长导出路的概念(即 LIP)及其优点,改善了基于 hypercube 拓扑结构的多处理机系统的容错性,并在此基础上给出求 LIP 的算法及实现;但是,由于该算法采用深度优先的搜索方式,因此复杂度较高,有待进一步改善。

参考文献:

[1] RAGHAVENDA CS, YANG PJ, TIEN SB. Free dimensions - an effective approach to achieving fault tolerance in hypercubes[A]. Proc. 22nd International Conference on Parallel Processing[C], 1993.

[2] SAAD Y, SCHULTZ MH. Topological properties of hypercubes[R]. Research Report 389. Dept. of Computer Science, Yale University, June 1985.

[3] 刘方爱,刘志勇,乔香珍. 光 RP(k) 网络上 Hypercube 通信模式的波长指派算法[J]. 软件学报, 2003,14(3).

[4] 高峰,李忠诚. 用最优通路矩阵实现超立方体多处理机系统的容错路由[J]. 计算机学报,2000,23(3).

[5] 王洪玉,董秀国,顾伟康. 全互连立方体网络的路由算法研究[J]. 浙江大学学报(工学版),2001,35(3).

(上接第 2110 页)

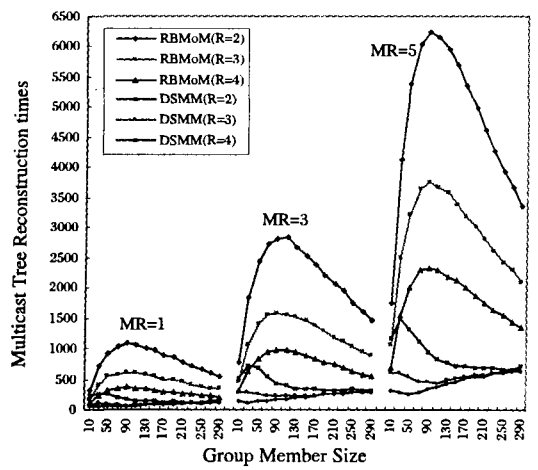

图 4　多播树维护代价的比较

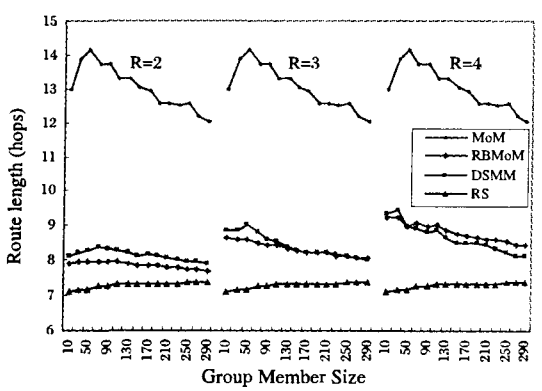

图 5　多播路由路径长度的比较(MR = 3)

多播数据的传输延迟可以用多播路由路径长度来表示。DSMM 使用了隧道转发多播数据,但并没有给多播数据带来太大传输延迟。图 5 给出了使用不同服务范围(R = 2,3,4)时多播路由路径长度的比较。从图中可以看出,DSMM 的路由路径长度介于 RS 和 MoM 两者之间,随着 DMSP 的服务范围增加,多播路由路径略变长。实验结果表明,DSMM 的多播路径长度 RBMoM 相当。结合图 3 和图 4 可以看出,DSMM 在没有增加太多传输延迟的情况下,却有效地减少了网络中的多播通信量,并降低了多播树的维护代价。

4　结语

通过分析和模拟实验表明,新提出的方案有效地减少了网络中的多播通信量,并降低了多播树的维护代价,同时没有给多播数据传输带来太大的延迟。

参考文献:

[1] PERKINSC. IP Mobility Support for IPv4[S]. RFC3344, Aug. 2002.

[2] CHI KH, TSENG CC, HUANG TL. A Framework for Mobile Multicast Using Dynamic Route Reconstructions[J]. Computer Journal, 1999, 42(6): 522 - 533.

[3] CHENG LT, PINK S. MobiCast: A multicast scheme for wireless networks[J]. Mobile Networks and Applications, ACM/Baltzer Mobile Networks and Applications, 2000, 5(4): 259 - 271.

[4] HARRISON TG, WILLIAMSON CL, MACKRELL WL, *et al.* Mobile multicast (MoM) protocol: Multicast support for mobile hosts [A]. Proceedings of ACM/IEEE MOBICOM'97[C], 1997. 151 - 160.

[5] LIN CR, WANG KM. Mobile Multicast Support in IP Networks [A]. INFOCOM2000[C], 2000. 1664 - 1672.

[6] YANG SJ, PARK SH. A Dynamic Service Range - Based Multicast Routing Scheme using RSVP in Mobile IP Networks[A]. IEEE Globecom2001[C], San Antonio, Texas, USA, Nov. 2001.

[7] PARK J, SUH YJ. An Efficient Multicast Routing Protocol For Mobile Hosts[J]. Proceedings of KISS Spring Conference, 2001 vol. 21: 262 - 264.

[8] LAI J, LIAO W. Mobile multicast with routing optimization for recipient mobility[J]. IEEE Transactions on Consumer Electronics, 2001, 47(1): 199 - 206.

[9] RIZZO L, VICISANO L. RMDP: An FEC-based reliable multicast protocol for wireless environments[J]. ACM Mobile Computing and Communications Review, 1998, 2(2): 23 - 31.