RCR 네트워크에서 최단경로를 위한 탐색 알고리즘

김성열

A Searching Algorithm for Shortest Path in RCR Network

Seong-yeol Kim

요 약

RCR 네트워크 토폴로지[1]는 짧은 지름, 대칭성 등의 특징을 가지고 있어 병렬컴퓨팅 환경을 구성하기에 적합한 상호접속네트워크의 일종이다. Hu and Cao[2]에 의하여 이 토폴로지 분석에 대한 재검토가 이루어졌으며, 그래프 비연결성, 직경, bisection width 등에 대한 오류가 있음을 지적하였다. 이 논문에서는 RCR 네트워크 토폴로지 특성을 분석하고, [2]의 결과에도 여전히 남아있는 '연결그래프가 되기 위한 조건' 및 직경에 대한 오류를 정정한다. 그리고 RCR 네트워크에서 최단경로를 구하기 위한 알고리즘을 제안한다.

ABSTRACT

RCR network[1] is a topology for interconnection networks having many desirable properties for building scalable parallel machines. This had been analyzed by Hu and Cao[2] to deal with problems of disconnected graph, bisection width and diameter. We analyze some properties of RCR again and revise the condition for connected graph and network diameter. And we present an efficient algorithm for finding next node on a shortest path.

키워드

RCR(Recursive Cube of Rings), Connected graph, Network diameter, Shortest Path

I. Introduction

In a parallel system, the interconnection topology plays an important role in determining the overall performance of the system. If the network cannot provide adequate performance, for a particular application, nodes will frequently be forced to wait for data to arrive. This networks include Mesh, Rings, Hypercube, X-Tree, Cube Connected Cycles[3], Butterfly[4], Shuffle Exchange, Shuffled-Mesh[5], HCC[6,7], etc. Among the various parallel interconnection topologies, a new family of scalable interconnection network topology named RCR (Recursive Cube of Rings)[1] was proposed. RCR(k, r, j) is the notation for a RCR where k is dimension of the cube, r is the number of nodes on a ring and j is the number of expansions from general seed. This topology has many desirable

울산과학대학 컴퓨터정보학부(green@mogent.net)
접수일자 : 2010. 09. 20 심사(수정)일자 : 2010. 10. 01 게재확정일자 : 2010. 10. 15

properties for building scalable parallel machines such as fixed degree, small diameter, high bisection width and symmetry. Even though this topology was nice, it made a little mistake in defining its properties. The work [2] gave three notices including disconnected graph, bisection width and network diameter. But we found the diameter defined in [2] is applicable to some *k, r, j* but not to others. The routing algorithm presented in [1] does not find a shortest path from source node to destination node.

In this paper we define the condition for RCR(*k, r, j*) to be a connected graph and network diameter of RCR(*k, r, j*). Then we propose an algorithm for finding next node to make a shortest path.

II. Condition for Being A Connected Graph

RCR(*k, r, j*) is a network which cubes are connected by rings[1]. Address of a node *n* is notated as $[A_n, b_n]$, where $0 \le b_n \le r-1$, $A_n = a_{k+j-1}a_{k+j-2}...a_i...a_0$, and $a_i \in \{0, 1\}$. Let a_i of A_n be $A_n(a_i)$ hereafter. In this network, each node has *k* cube-links and two ring-links when $r > 2$, belonging to a *k*-cube as well as a ring. $[A_m, b_m]$, a cube-neighbor of $[A_n, b_n]$, has address in which only one bit $A_m(a_i)$ is different from $A_n(a_i)$, where $i = -bj-x \ (mod\ k + j)$, $1 \le x \le k$. Two ring-neighbors are $[A_n, b_n \pm 1 (mod\ r)]$.

Hu[2] described that RCR(*k,1,j*) is a disconected graph. However, many cases being disconncted on RCR(*k, r, j*) exist. One of them is from [00000, 1] to [00100, 1] on RCR(2,2,3). We now define the condition for RCR(*k, r, j*) being a conected graph.

Theorem 1: RCR(*k, r, j*) is a connected graph , iff $k(r-1) \ge j$.

Proof: According to generation of RCR(*k, r, j*), node *n* has (*k*+2) neighbors containing *k* cube-links and two ring-links when $r > 2$.

$[A_m, b_m]$, one of *k* cube-neighbors of $[A_n, b_n]$, is different from *n* by only one bit a_x, where $A_n(a_x) \neq A_m(a_x)$, $x \in X$, $X = \{-bj-1, -bj-2, ..., -bj-k \ (mod\ k+j)\}$ and $b_n = b_m$. These *k*+1 nodes, node *n* and it's *k* cube-neighbors, make a cube together with other 2^k-k-1 nodes which have same addresses except a_x, $x \in X$. This means that these 2^k nodes are connected by $k \times 2^{k-1}$ cube-links.

But if we want to connect from node n to a node d where $A_d(a_y) \neq A_n(a_y)$, $y \notin X$, we must move through a ring-link. $[A_t, b_n+1]$, a ring-neighbor of $[A_t, b_n]$ which is a part of the cube including $[A_n, b_n]$, has cube-neighbors that have the addresses different from A_t by only one bit ay where $y \in Y$, $Y = \{-(b+1)j-1, -(b+1)j-2, ..., -(b+1)j-k\}$. Similarly, these (*k*+1) nodes make a cube together with other 2^k-k-1 nodes which have same addresses except a_y, $y \in Y$,

where $Y = \{-(b+1)j-1, -(b+1)j-2, ..., -(b+1)j-k(mod\ k+j)\}$
$= \{-bj+k-1, ..., -bj+1, -bj(\text{mod } k+j)\}$

From this fact, we find that Y has k consecutive bit positions to $(-bj-1)$ position in X. Therefore node n can be connected to any arbitrary node if and only if $k \times r$ is greater or equal to $k + j$.

III. New Definition of Network Diameter

Hu[2] defined network diameter D= $k+j+br/2c+1$. But in RCR(2, 7, 3), distance from [00000, 0] to [11111, 5] is 10, not 9. One of feasible shortest path is as follows. [00000, 0]→ [10000, 0]→[11000, 0]→[11000, 1]→[11001, 1]→ [11011, 1]→[11011, 2]→ [11111, 2]→[11111, 3]→ [11111, 4]→[11111, 5]

Here, distance from [00000, 0] to [11111, 2] is 7, where 5 cube hops plus 2 ring hops. Later, the path from [11111,2] to [11111,5] is traversed only on ring side. Now we define network diameter.

Theorem 2: For a connected RCR(*k, r, j*), network diameter D is $k + j + [j/k] + [r/2]$.

Proof: In order to move from $[A_c, b_c]$ to $[A_d, b_d]$,

we first move from [A_c, b_c] to [A_d, t], then to [A_d, b_d]. When $A_d = \overline{A_c}$, that is $\overline{A_c}(a_i) \neq A_c(a_i)$, $0 \leq i \leq k+j-1$ the first movement needs $k+j+x$ hops, $k+j$ cube hops and x ring hops, as described in Theorem 1. Here, x is $[(k+j)/k]-1$ at maximum, since more k bits can be covered when we use one ring link. So t can be $b+x$ or $b-x$ depending on the direction traversed on rings.

In order to reach from [A_d, t] to [A_d, b_d], a path is traversed from t to b_d on a ring. It requires maximally $[r/2]$.

Therefore the maximum distance between two nodes is $k+j+x+[r/2] = k+j+[j/k]+[r/2]$.

IV. Algorithm for Finding Next Node on A Shortest Path

Sun[1] proposed a routing algorithm and insisted the algorithm can find a shortest path from [A_c, b_c] to [A_d, b_d]. According to the algorithm, in case of $A_c = A_d$ and $b_c \neq b_d$, next node would be [A_c, b_v] such that $|b_v - b_d| \leq |b_c - b_d|$. The selection of next node is wrong in some cases. For example, if [A_c, 1] is the source node, [A_c, 5] is the destination node and $r = 6$, then choice of [A_c, 0] makes a shorter path than [A_c, 2], while $|2-5| \leq |1-5|$.

We propose an efficient routing algorithm that guarantee a shortest path. This algorithm is very simple and clear. The function *FindingNextNode* is to find next node to make a shortest path. Let''s consider the connection from [A_c, b_c] to [A_d, b_d].

In case that $A_c = A_d$ and $b_c \neq b_d$, the path is traversed only on a ring. So we have to choose either b+1 or b−1 which is closer to b_d. In our algorithm, the function *DistanceOnRing*(b_c, b_d) returns the distance from b_c to b_d on a ring and the ring size r is declared as a global variable.

In other case that $A_c \neq A_d$, we transfer through a cube link. However, if there is no cube link closer to destination, we should choose a ring link. The function *FindCubeNeighbor*([A_c, b_c], [A_d, b_d]) returns a cube-neighbor of [A_c, b_c] for connecting to [A_d, b_d]. If this function returns [A_c, b_c], this means that there is no cube link. For this case, we choose a ring link closer to [A_d, b_d].

The function *Distance*([Ac, b_c], [A_d, b_d], *direction*) computes the distance from [A_c, b_c] to [A_d, b_d] depending on a specific direction, where (*direction* = +1) means that the path moves to forward direction on a ring and (*direction* = −1) means to backward direction on a ring. We compare the cost produced by traversing to forward direction and to backward direction and then choose small one. The cost is determined by addition of two costs, one is the cost from [A_c, b_c] to [A_d, t] and the other is the cost of from [A_d, t] to [A_d, b_d]. If (*direction* = +1), then t will be b_c+x, otherwise, t will be b_c-x', where x (or x') is ring hops that needs to travel from [A_c, b_c] to [A_d, t]. The function *Distance* first calculate cube hops by executing $A_c \oplus A_d$. After that, ring hops required to traverse from [A_c, b_c] to [A_d, t] is added to this cubehops in the while-block. Later, the distance from [A_d, t] to [A_d, b_d] on a ring is added to compute total cost.

```
global varibles : int k, r, j
                  // k, r, j : come from RCR( )
node FindingNextNode(node[Ac,bc],node[Ad,bd]) {
//[Ac,bc]:current node, [Ad,bd]:destination node,
//[An,bn]:next node
if (Ac=Ad and bc=bd) return [Ac, bc];
if (Ac=Ad and bc≠bd) { d1 = DistanceOnRing(bc
+ 1, bd);
                d2 = DistanceOnRing(bc−1, bd);
                if (d1≤d2) bn=bc+1 else bn= bc−1
                return [Ac, bn]; }
if (Ac≠Ad ) {
    [An,bn] = FindCubeNeighbor([Ac,bc], [Ad,bd]);
// finding a cube neighbor lead to destination
    if(Ac≠An) {bn=bc return [An, bn]; }
```

```
d1 = Distance([Ac, bc], [Ad, bd] , +1 );
 // the case of travelling forward on a ring
d2 = Distance([Ac, bc], [Ad, bd] , -1);
 // the case of travelling backward on a ring
if (d1≤d2) bn = bc + 1 (mod r);
    else bn = bc −1 (mod r);
return [An, bn];
} }
```

Ⅴ. Conclusion

As a interconnection network topology for building scalable parallel machine, RCR network, has many desirable properties such as fixed degree, small diameter, high bisection width and symmetry.

RCR topology was proposed by Y.Sun[1] and discussed again by H.Hu[2]. We analyzed some properties of RCR network, made a correction for some mistakes that still remains, and proposed an algorithm for finding next node for the destination to make shortest path.

References

[1] Y.Sun, P.Y.S.Cheung and X.Lin, ""Recursive Cube of Rings: New Topology for Interconnection Networks"", IEEE Trans. Parallel and Distributed System, vol.11, no.3, pp.275-286, 2000.

[2] H.Hu, N.GU, and J.Cao, ""A Note Recursive Cube of Rings Network"", IEEE Trans. Parallel and Distributed System, vol.16, no.10, pp.1007- 1008, 2005.

[3]]F.P. Preparataand J.E. Vuillemin, The cube-connected cycles: A versatile network for parallel computation, in: Proc. of the 12th Annual IEEE Symposium on Foundations of Computer Science, pp.140-147, 1979.

[4] S.N. Bhatt, F.R.K. Chung, J.-W. Hong, F.T. Leightonand A.L. Rosemberg, Optimal simulations by butterfly networks, in: Proc. of the 20th AnnualACM Symposium on the Theory of Computing, pp.192-204, 1988.

[5] G. Bongiovanni, G.A. De Biase, A. Massiniand A. Monti, *The ShuffledMesh: a flexible and efficient model for parallel computing*, Telecommunication Systems v63.13 pp.21-27, 2000.

[6] Toshinori Takabatake, Keiichi Kaneko, Hideo Ito, "Generalized Hierarchical Completely-Connected Networks", International Symposium on Parallel Architectures, 1999.

[7] Toshinori Takabatake, Tomoki Nakamigawa, Hideo Ito, “Connectivity Of Generalized Hierarchical Completely-Connected Networks”, Journal of Interconnection Networks, Vol.9, no.1, pp. 127-139, 2008.

APPENDIX

FUNCTIONS USED IN OUR ALGORITHM

global varibles : *int k, r, j* ;

$node$ FindCubeNeighbor (

$node[A_c,b_c], node[A_d,b_d]$) {

$A_n = A_c; A_t = A_c \oplus A_d$;

for $(i=1;\ i \le k;\ i{+}{+})$

$y = -(b \times j) - i(\mathrm{mod}\ \ k+j)$;

$If(A_t(y)=1)\ \mathrm{A_n(y)} = \overline{\mathrm{A_n}}(\mathrm{y})$;

$break;\ \}\}$

return $[A_n, b_c]$;

}

int Distance $(node[A_c,b_c], node[A_d,b_d]$

$, int\ direction$) {

$A_t = A_c \oplus A_d$;

cubeHopSum = the number of bits of 1 in A_t

d = cubeHopSum

while ( ) {

cubeHops = $\sum_{jj=1}^{jj=k} A_t[b_c \times k - jj(\mathrm{mod}\ k+j)]$;

if (cubeHops ≥ 1)

```
        cubeHopSum = cubeHopSum- cubeHops;
    if ( cubeHopSum == 0 ) break;
    d = d + 1;
  b_c = b_c +  + direction (mod r)
}
return  d + DistanceOnRing(b_c, b_d) ;
}
int  DistanceOnRing ( int  b_c , int  b_d  ) {
    d = |bc − bd|  ;
    if (d ≤ ⌊ r/2 ⌋ ) return  d;
          else return ( r − d);
}
```

■

저자 소개

김성열(Seong-yeol Kim)

1994년 조선대학교 전자계산학과(이학사)
1996년 조선대학교대학원 전자계산학과(이학석사)
2000년 조선대학교대학원 전자계산학과(이학박사)
2002년 ~ 현재 울산과학대학 컴퓨터정보학부 부교수
※ 관심분야 : 정보보안, 분산시스템, 무선인터넷, 가상화, 임베디드시스템, 클라우드컴퓨팅