Deadlock-free wormhole routing algorithms for star graph topology

C.P.Ravikumar
A.M.Goel

Indexing terms: Parallel processing, Routing algorithms, Routing optimisation

Abstract: For constructing massively parallel multicomputers with over 5000 processing nodes, the Star Graph topology is known to be better than the hypercube in terms of the average routing distance, the number of links per node, and the fault diameter. The authors present two deadlock-free algorithms for routing in Star Graph, assuming the Wormhole routing model. Both the algorithms use the concept of virtual channels introduced by Dally and Seitz. The first algorithm is non-optimal in terms of the average routing distance, but uses fewer virtual channels on the whole. The second algorithm is optimal in terms of routing performance, but requires a somewhat larger number of virtual channels per node.

1 Introduction

Message-passing multiprocessors based on direct interconnection networks have become popular in building massively parallel systems with over 5000 nodes. The hypercube interconnection network, for instance, has been used in many existing commercial parallel processors such as the Intel iPSC/860 [7], NCUBE/7 [3], and the Connection Machine [4]. More recently, Akers and Krishnamurthy [8] introduced a class of graphs, known as Cayley Graphs, on which communication-efficient interconnection networks with good fault-tolerance properties can be constructed. A particular subclass of Cayley Graphs, known as Star Graphs, are known to outperform the hypercube in terms of the average distance, the number of links per node, and the fault diameter when the number of processing nodes is over 5000 [8]. An n-star graph consists of *n*\ nodes labelled using permutations of 1 2 ... *n.* Two nodes labelled *s* and *t* are joined by an edge iff *t* can be obtained by swapping the first (left most) symbol of *s* with any other symbol of *s.* For instance, a 4-star has 24 nodes; the node labelled 1234 is connected to 3 other nodes - 2134, 3214, and 4231. Fig. 1 shows the topology of a 4-star. If the label *t* can be obtained by swapping the z'th symbol of *s* with the first symbol of *s,* we write

$$t = 5«(«)$$

where g, is the rth 'generator' function which simply swaps the symbol s_t, with s_f. The following properties of the Star Graph were proved by Akers and Krishnamurthy [8].

(a) Star Graphs are node-symmetric and edge-symmetric.

(b) The degree of a node in an «-star is *{n* - 1)

(c) The H-star is maximally fault-tolerant, with a fault-tolerance of *(n* - 2).

(d) The communication diameter of the «-star is L3(H - l)/2j.

(e) The average distance between two nodes in an *n*-star is $n - 4 + H_n + 2/n$, where H_n is the nth Harmonic number.

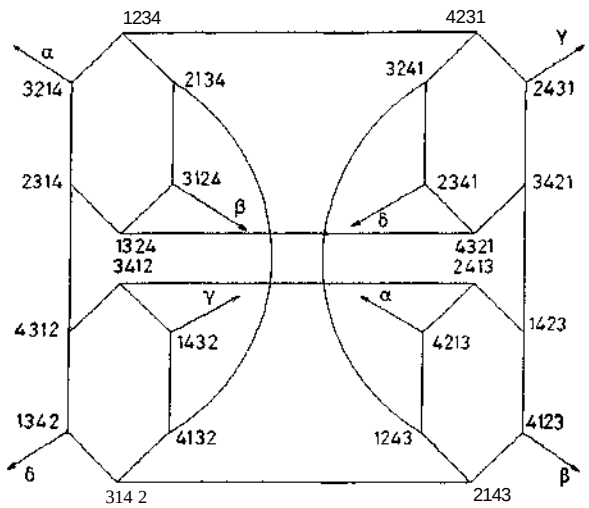

Fig. 1 *4-star topology*

7.7 *Wormhole routing*

Inspired by computer networks, early multicomputers made use of store-and-forward routing technique, where the entire message is buffered at each intermediate node before the message is forwarded to the next node in the path [6]. In store-and-forward routing, the communication delay is large and is directly proportional to the length *L* of the path along which the message is routed. Dally and Seitz [2] introduced a routing technique called Wormhole routing, which overcomes the above disadvantage. In a Wormhole-routed network, each message packet is subdivided into flow control digits (flits). Flits are routed from the source to the

IEE Proceedings online no. 19952221
Paper first received 13th October 1994 and in revised form 6th July 1995
The authors are with the Department of Electrical Engineering, Indian Institute of Technology, New Delhi 110016, India

destination in a pipelined fashion. Thus, the header flit can arrive at the destination even before the last flit of the message has left the source. All flits that belong to a message packet make use of the same set of communication channels. In Wormhole routing, the communication delay depends mainly on the bandwidth of the communication channels. Several commercial machines such as the Intel iPSC/860 and research prototypes such as the J-machine have adopted the Wormhole routing technique [6]. A disadvantage of Wormhole routing is that deadlocks can occur if special precautions are not taken to avoid deadlocks by carefully restructuring the routing algorithm [2]. Dally and Seitz presented deadlock-free routing algorithms for several network topologies such as the fc-ary «-cube, the Shuffle-Exchange network, and the Cube-connected Cycles [2]. In this paper, we present two deadlock-free algorithms for the Star Graph topology.

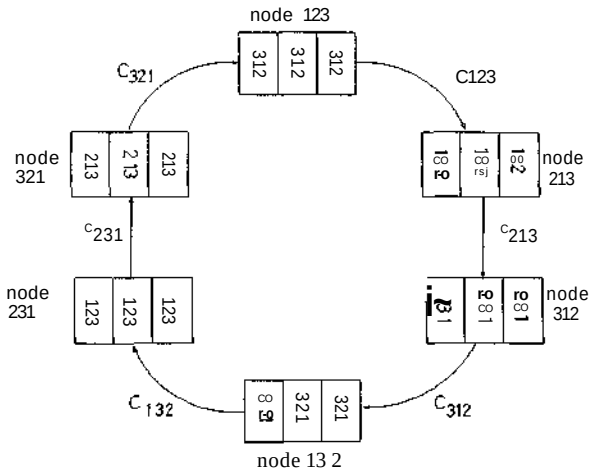

Fig.2 *Deadlock situation in a 3-star*

A routing algorithm for the K-star, which takes no more than L3(« - l)/2j steps to route a message from any given node S to a destination node D, was given in [8]. However, the Akers-Krishnamurthy routing algorithm is not deadlock-free, as can be seen in the example of the 3-star (see Fig. 2). We label the channels of the 3-star c_s, where *s* is the label of the source node of the channel. Using the Akers-Krishnamurthy algorithm, the node 213 must forward the flit received along channel c_{123} onto the channel c_{132}. Following [2], we construct a channel dependence graph G which has one node corresponding to each channel. A directed edge exists from a node c_i in G to a node c_j if a flit is presently on channel c, and the routing algorithm would send the flit onto the channel *Cj* in the next step. If each channel is associated with a flit-buffer of size 3, a deadlock situation occurs in this example, where no flit can progress towards its destination. Dally and Seitz showed that a routing algorithm is deadlock-free if and only if the channel dependence graph G induced by the routing algorithm does not contain any directed cycles. We shall ensure this by restricting the routing such that a directed edge exists in the channel dependence graph from channel c_i to channel *Cj* only if $i > j$.

1.2 Organisation

In the following section, we present a routing algorithm for the K-star that is similar to the e-cube algorithm for Hypercube [2]. While the e-cube algorithm is deadlock-free, our e-star algorithm is not; we show how the *e*-star algorithm can be rendered deadldck-free [5] using the concept of virtual channels introduced by Dally and Seitz. However, as we shall prove in Section 2, it turns out that the e-star algorithm is not optimal in terms of the average routing distance. In Section 3, we present a 'cycle merging' routing algorithm which is optimal and deadlock-free. The two algorithms are compared and conclusions are drawn in Section 4.

2 The e-star algorithm

The e-cube routing algorithm for a binary K-cube routes a message originating at source S by 'correcting' one bit at a time, starting from the most significant bit of S. For instance, a message originating at 10110 and intended for the destination 01101 in a 5-cube will be routed along the path 10110-00110-01110-01100-01101. The e-cube algorithm, which can also be generalised for the *k-ary* «-cube, is deadlock-free [2]. We can derive a similar algorithm for routing in the n-star. This algorithm, which we call the e-star, corrects the source label S one position at a time, starting at the rightmost position. As an example, a message originating at node 2341 and intended for destination node 1234 will be routed along the path 2341-4321-1324-3124-2134-1234.

```
procedure e-star (s, D, n, M);
begin
/* Route message M from node S to node D in an n-star */
SO:  if S = D then consume M;
     else begin
SI:    Find largest i <= n such that D(z) =t S(z)
       if S(l) = D(z) then begin
S2:      T := g,(S);
         e-star (T, D, n, M);
       end else begin
S3:      Find 7 such that S(/) = D(z);
         T := g/S);
         e-star (T, D, n, M);
       end
  end
end
```

The e-star algorithm may be understood using the concept of sub-stars in a star graph. In the n-star, consider the subgraph induced by all those nodes whose labels have the symbol *i* fixed in their *jth.* position; we denote this subgraph by *ij*. For instance, 2_4^3 represents the subgraph induced by the nodes 1324, 3124, 1423, 4123, 3421, 4321. It is easy to see that *ij* is a sub-star when $j > 1$ [8]. Given a node *s*, we denote the sub-star induced by keeping *s(J) s(j* + 1) ... *s(ri)* fixed in their positions by *y(Sj),* $1 < j \leq n$. It is easy to see that \|/(^) is also a sub-star.

The basic idea behind the e-star algorithm is to check if the source S and destination D are on the sub-star \]/(D_n); if not, the algorithm routes the message to a node which is on the sub-star \|/(D_n). Thus the algorithm has 'corrected' the last symbol of S to match the last symbol of D. This procedure is repeated, until all the symbols of S are corrected to match the symbols of D.

Theorem 1: The e-star algorithm correctly routes a message from a source S to destination D.

Proof: Given the label of the current node S and the destination node D, we define the function *f[S,* D) to be the largest index *i* such that S(z') *t-* D(z). When S = D, the function/is defined to be 0. Suppose that the *e*-star algorithm routes the message from a node P to node Q. Then it is necessarily true that *f(Q,* D) $\leq$ /(P, D), where D is the destination node. This may be seen by looking at Steps SI and S3 in the above algorithm. Step SI evaluates *i* = *f(S,* D). If the algorithm takes Step S2, T is computed to be g,(S), in which case T(z) = D(r'). Furthermore, the generator g, only changes the z'th and 1st symbols of S. Therefore, /(T, D) is at least one less than *f(S,* D). If the algorithm takes Step S3, then /(T, D) = *f[S,* D) since 1 < *j* < *i.* We now claim that whenever the e-star algorithm takes the Step S3, the following call of e-star will take Step S2. If T is the node to which Step S3 routes the message, then T(l) = S(/) = D(i). In the following call of e-star, T takes on the role of S, and hence S(l) = D(/), forcing the algorithm to execute Step S2. Further, Step SI ensures that the algorithm will only stop forwarding the message if /(S, D) = 0. Hence the proof.

Theorem 2: The e-star algorithm takes no more than *In* - 3 steps to route a message on an «-star. The average routing distance achieved by the algorithm is 2« + 1 - m_n.

Proof: For correcting any position in the permutation corresponding to S, the e-star algorithm requires either no forwarding, a single forwarding step (Step S2), or two forwarding steps (Step S3 followed by Step S2). In the worst case, the algorithm takes 2 forwarding steps to correct each of the rightmost *(n* - 2) symbols, and then uses the Step S2 to correct the second symbol. Let T(«) be the average routing distance between any two nodes S and D on an «-star. Using the e-star algorithm, routing on an n-star is equivalent to correcting the rightmost position and then routing on the *n* - 1 sub-star. The symbol required to correct the rightmost position can be in any of the *n* positions of S with equal probability. If this symbol is in the first position, one forwarding step is required to correct the rightmost position. If the symbol is already in the rightmost position, no forwarding step is required to route to the sub-star. Finally, if the symbol is in any other position *j,* 1 < *j* < *n,* two forwarding steps become necessary (Step S3 followed by Step S2). Hence, the following recurrence relation results for T(«).

$$T(n) = \left(\frac{1}{n} \times 0 + \frac{1}{n} \times 1 + \frac{n-2}{n} \times 2\right) + T(n-1) \quad (1)$$

It is easy to see that the basis T(2) = V_2. Solving the recurrence gives the required formula for T(n).

Theorem 3: The e-star algorithm of Fig. 3 is not deadlock-free.

Proof: For the routing pattern of Fig. 2, the e-star algorithm behaves identically as the Akers-Krishnamurthy algorithm, thereby inducing a directed cycle in the channel dependence graph.

2.7 Making the e-star deadlock-free

We use the concept of virtual channels introduced by Dally and Seitz [2] to achieve deadlock-free e-star routing. Each communication channel in the «-star is divided into virtual channels; these virtual channels share the same physical channel, but use independent flit buffers. The notation used in this section is shown as follows

x, y, z	permutations on *n* symbols
n_x	node labelled using permutation *x*
sym(y, *d)*	*dth* symbol of permutation *y*
sJj)	permutation obtained by applying generator g_d to *y* i.e., by changing sym(y, 1) and sym(j, *d)*
^{c}ky	physical channel from node n_y to $n_{gn \cdot k(y)}$
n_s	Source node
n_D	Destination node
R(c, n_z)	Routing function *R* returns the channel along which a message received on channel *c* and destined for node n_z will be routed.

We split the physical channel c_{ky} into *(k* + 1) virtual channels. A virtual channel is labelled c_{vky}, where v indicates the virtual channel number *(n* - *k)* $\leq$ v $\leq$ *n.* Thus c_{vky} is the virtual channel labelled v and shares the physical channel c_{ky} which in turn connects node n_y to $n_{gn \cdot k(j,y}$ We modify the e-star algorithm as shown below; for convenience, we use *d* = *(n* - *k).*

$$R_{e-star}(c_{v,k,y}, n_z) = c_{d',n-d',g_d(y)}$$

if Vj > *d!* $sym(g_d(y),j) = sym(z,j)$

$\wedge$ $sym(g_d(y),d') \neq sym(z,d')$

$\wedge$ $sym(g_d(y),l) = sym(z,d')$

$= C_{d',n-i,gd(x)}$

if Vj > *d'* $sym(g_d(y),j) = sym(z,j)$

$\wedge$ $sym(g_d(y),d') = sym(z,d')$

$\wedge$ $sym(g_d(y),i) = sym(z,d')$

$\wedge$ $i \neq 1$

We refer to two equations above as Rules 1 and 2 respectively. Thus both the rules use the J'th virtual channel for routing, where *d* is the rightmost position in which the current node differs from the destination. Rule 1 is the case when the required symbol is in the first place, and Rule 2 is the complementary case.

Theorem 4: The routing function R_{e-star} is deadlock-free.

Proof: It is enough to show that the routing algorithm *Re-star* routes in the decreasing order of channel subscripts. Let c_{ykx} be the channel along which the message is received, and let $cv',k',g_{n-k(x)}$ be the channel along which the message is routed. We intend to show that

$$v\ k\ x > v'\ x'\ g_n{\sim}k(x)$$

If channel c_{vXx} was opted for by Rule 1, then we are assured that *v* > *v'* since v = *f{x, z)* and *k* = *n* - v; therefore,

$$v' = f(g_{n-k}(x),z) = f(g_v(x),z) < v$$

If channel c_{vkx} was chosen by Rule 2, then we are assured that v = v' and *k* > *kf* because v = *fix, z), n* - *k* < *v* and therefore

$$v' - f\{g_{n-k}(x),z) = f(x,z)$$

since $(n - k) < v$. This further implies that v = v'. Also, since Rule 2 is followed by Rule 1, we have $(n - V) = v'$.

2.2 Number of virtual channels

Theorem 5; It is sufficient to split the physical channel c_{ky} into $(k + 1)$ virtual channels, i.e., no more virtual channels are necessary.

Proof: When a packet intended for node n_z is routed along c_{vkx}, we have/(x, z) = v. Rule 1 is used if sym(x, 1) = sym(z, v) and we use v = $(n - k)$. Rule 2 is used if sym(x, i) = sym(z, v) for $i > 1$; in this case, we use $i = (n - k)$. We claim that when Rule 2 is used $2 \leq i < v$. This is because, for $j > v$, sym(x,y) = sym(z, j). Thus, in either case, $(n - k) < v$. Therefore, we divide the physical channel c_{ky} into $(k + 1)$ virtual channels labelled $c_{v,k,x'}$ $(n-k)<v<n$.

3 Cycle-merging algorithm

Is well known that a permutation P on n symbols can be represented by its cycle structure [1]. In this section, we extend the concept of cycle structure slightly and define the cycle structure of a permutation P with respect to a permutation Q. If $P = p_1 p_2 \cdots p_n$ and $Q = q_1 q_2 \cdots q_n$ then the cycle structure of P with respect to Q is obtained by constructing the following graph on n nodes labelled from 1 to n. A directed arc is drawn from node i to node j if $i = p_x$ and $j = q_x$ for some x. The cycles in this graph are then written down in some order. The reader will observe that our definition boils down to the conventional definition of cycle structure when Q is the identity permutation.

The essential idea behind the cycle merging algorithm is to transform the source permutation S into the destination permutation D. To do this, we write down the cycle structure of S with respect to D. The source S would have been transformed to D when this cycle structure consists of n unicycles. We achieve this in two phases.

[8] Let there be u unicycles in the cycle structure of S with respect to D. We leave the unicycles untouched. We consider the remaining cycles (of length larger than 1) and 'merge' them into one single cycle of length $(n - u)$. A special case occurs if the first symbol of S forms a unicycle. In this case, we merge this unicycle as well and obtain a merged cycle of length $(n - u + 1)$.

• The cycle of length $(n - u)$ or $(n - u + 1)$ in the special case mentioned above, is now reduced to $(n - u)$ or $(n - u + 1)$ unicycles through a series of swaps.

Example 1: Let S = 1 2 3 5 4 6 and D = 4 5 1 2 3 6. The cycle structure of S with respect to D is (1 4 3) (2 5) (6). Here $u = 1$ and we merge the remaining two cycles by swapping symbols 1 and 2. The resulting cycle structure is (2 4 3 1 5) (6) and the permutation is 2 13 5 4 6. Now we use a sequence of 4 swaps as shown in Table 1 to transform S to D.

Table 1

Swap	Permutation	Cycle structure
2 and 5	5 13246	(4 3 1 5) (2) (6)
5 and 1	15 3 2 4 6	(4 3 1) (5) (2) (6)
1 and 3	35 1246	(4 3) (1) (5) (2) (6)
3 and 4	45 1236	(4) (3) (1) (5) (2) (6)

The cycle merging algorithm can be more formally stated as follows,

```
procedure Cycle-Merge(S, D)
begin
  Phase-1: Find least i such that S[z] ^ D[z] and S[i] is
    not in the same cycle as D[1].
      If no such i can be found then go to Phase-2.
      Swap S[z] with S[1].
      Go to Phase-1.
  Phase-2: Find i such that S[1] = D[z]
      if i = 1 then stop.
      Swap S[1] with S[i].
      Go to Phase-2.
end
```

Theorem 6: At the end of the procedure Cycle-Merge, S is transformed to D.

Proof: Let y indicate the set of all cycles of length greater than 1 in the cycle structure of S with respect to D. Denote by 8 the cycle which contains the first symbol of D. (8 may or may not be a unicycle.) If y contains at least one cycle other than 8, Phase-1 will merge one of these cycles with 8. Phase-1 is repeated until y contains only 8. Thus, at the end of Phase-1, the cycle structure of S with respect to D consists of 8 and some unicycles, if any. We now claim that when Phase-2 completes, there are n unicycles in the cycle structure of S with respect to D. This follows from Lemma 1 given below, and the fact that the swap statements are executed until 8 becomes a unicycle.

Lemma 1: The swap statement in Phase-2 creates exactly one unicycle and reduces the length of 8 by 1.

Proof: Both S[1] and S[ż] in the swap statement belong to 8, since S[z] and D[ż] belong to the same cycle and S[1] = D[ż]. After the swap, a unicycle containing D[z] is created and the length of 8 reduces by 1. See Fig. 3.

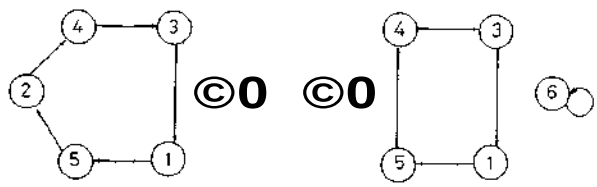

Fig.3 5 *before and after the swap statement in Phase-2: (a) before swapping; (b) after swapping*

Theorem 7: The procedure Cycle-Merge uses the optimal number of swap statements to transform S to D.

For a permutation Y on n nodes, let $p(Y, D)$ denote the number of cycles of length greater than 1, excluding the cycle containing D[1]. Let $q(Y, D)$ denote the number of symbols of Y that are not in their proper position with respect to D, excluding the symbol D[1]. Define

$$r(Y,D)=p(Y,D)+q(Y,D)$$

It is clear that r(Y, D) = 0 if and only if Y = D. The algorithm Cycle-Merge transforms the given source S to D by reducing r(S, D) to 0 through a sequence of swaps. Table 2 shows all possible moves in a star graph and their effect on r(Y, D). In this table, N indicates a symbol in Y that is in its correct position with respect to D; similarly, M indicates a symbol in Y that is not in its correct position with respect to D. From Table 2,

Table 2: All possible moves in a Star Graph

Move type	Y[1]	Rule	1st Symbol of new Y	Ap(Y,D)	Aq(Y,D)	Δr(Y,D)
1	D[1]	Swap with any N	X	0	1	1
2	D[U	Swap with any M	**X**	-1	0	-1
3	D[i'/],;> 1	Swap with *S[i]*	D[1]	0	-1	-1
4	D[*i], i'#* 1	Swap with S[*/*]	X	0	-1	-1
5	**D[7**], /V 1	Swap with any N	**X**	0	0	0
6	D[*J], i** 1	Swap with any M in same cycle	X	0	0	0
7	**D[***i*\, /* 1	Swap with any M in different cycle	X	-1	**0**	-1

it is clear that no move in a Star Graph can reduce r(Y, D) by a value larger than 1. Therefore, for any given source S, *r(S,* D) is a lower bound on the number of moves required to transform S into D. The algorithm Cycle-Merge makes only those moves which reduce *r(Y,* D) by 1. Specifically, the algorithm avoids moves 1, 5 and 6 (see Lemma 2). Hence the theorem.

Lemma 2: The algorithm Cycle-Merge never uses moves of type 1, 5 or 6.

Proof: During Phase-1, the first symbol of S is swapped with a symbol that is not a part of a unicycle. Thus moves 1, 5 are not taken during Phase-1. Further, the symbol chosen for swapping with S[l] must be in a cycle not containing S[l] and D[1]. Therefore, moves 3, 4, 6 are not chosen by Phase-1. In Phase-2, the symbol S[l] is taken to its correct position with respect to D. Therefore, only moves 3 and 4 are permissible in Phase-2.

3.1 Performance

We now prove some performance results in connection with the Cycle-Merge algorithm. It will be seen that the algorithm provides optimal performance in terms of routing delays.

Theorem 8: The maximum number of swaps made by the algorithm Cycle-Merge is _3(n - 1)/2J. The average number of swaps x(«) is given by (« + *2/n* + H_n - 4).

Proof: In Theorem 7, we have seen that the routing distance between source S and destination D is given by r(S, D) = *p(S,* D) + *q(S,* D). An upper bound on *p(S,* D) is [(« - l)/2j since we can have at most _*(n* - 1)/2J cycles of length larger than 1 on (« - 1) symbols. It is easy to see that an upper bound on *q(S,* D) is *(n* - 1). Hence the maximum number of moves made by the algorithm Cycle-Merge is [_3(« - 1)2_J. In order to measure the average routing delay *x(n),* we develop a recursive formulation for *x(n).* Consider a permutation S' on (« - 1) symbols. We obtain a permutation S on *n* symbols by inserting the «th symbol into any one of the cycles of S'. Let *x(n* - 1) be the average number of routing steps associated with S'. The following cases arise.

(a) The rath symbol forms a unicycle in S. Clearly, *x(n)* = *x(n* - 1) in this case.

(b) The «th symbol is part of the cycle containing the first symbol. Here, *x(ri)* = *x(n* - 1) + 1 since one extra move will be made during Phase-2 to take «th symbol to its correct position.

(c) The wth symbol is inserted into a unicycle, resulting in a cycle of length 2. Here, *x(n)* = T(« - 1) + 3 since one extra move will be required in Phase-1 to merge the 2-length cycle and two Phase-2 steps will be required to take the two relevant symbols to their correct positions.

(d) The «th symbol is inserted into a cycle of length larger than 1 and the cycle does not contain the first symbol. In this case, *%{n)* = *%{n* - 1) + 1 since an extra move will be required during Phase-2 to put the «th symbol in its correct position. The average *x{n)* associated with S can be now worked out as

$$r(n) = p_1\tau(n-1) + p_2(\tau(n-1)+1) + (p_3\tau(n-1)+3) + p_4(\tau(n-1)+1) \quad (2)$$

where p_t is the probability of case *i,* $1 \le i \le 4$. It is easy to see that *p*\ = \l*n* Further, /?$_3$ is given by

$$P3 = \frac{\text{Average Number of Unicycles excluding (1) in S'}}{n} = \left(1 - \frac{1}{n-1}\right)\frac{1}{n}$$

We further use $p_2 + p\$ = 1 - (p\ + Pi)$ in eqn. 2 to obtain the recurrence relation for *x(ri).*

$$T\{TI) = r(n-1) + 1 + \frac{3}{n} - \frac{2}{n-1} \quad (3)$$

$$\tau(2) = \frac{1}{2} \quad (4)$$

Solving the above recurrence gives the required result.

The reader will observe that the Cycle-Merge algorithm gives the same (optimal) performance as the Akers-Krishnamurthy algorithm of [8].

3.2 Rendering the CMA deadlock-free

We use the concept of virtual channels defined by Dally and Seitz [2] to achieve deadlock-free routing. Each physical channel c_{dY} which originates from a node labelled *Y* and which connects to the node labelled *g/Y)* is implemented as a set of *(n* - 1) + \n - *dll*\ virtual channels. Specifically, we divide each physical channel into

• \n - *dll*\ cycle-merging channels denoted by $c_{Xkd Y}$ where _ \≤*k*<\n-*dl2*\.

• (n - 1) cycle-reducing channels $c_{Otkidi Y}$, \ ≤ *k* ≤ *(n - Y).* The following rules are used for routing.

(a) If Phase-1 of the Cycle-Merge algorithm has to be applied, then route along the cycle-merging channel ^{c}\,*k,d,Y,* where *k* = *p(Y,* D).

(b) If Phase-2 of the algorithm is to be used, then route along the cycle-reducing channel c_{okdY} where *k* = *q(Y,* D).

Formally, the routing function R_{CMA} for the cycle-merging algorithm may be defined as shown below.

$$
\begin{aligned}
R_{CMA}(c_{b)U},d,Y,D) \;=\; & c_{1iM,j5d(Y)} \quad if\ k=p(g_d(Y),D) \;>0 \\
& /\backslash\ \forall j < d!\ (sym(g_d(Y),j)\ \text{and}\ D[1] \\
& \text{are in same cycle in } g_d(Y) \text{ w.r.t. } D \\
& \forall \quad sym(g_d(Y),j)=-D[j]) \\
& \exists' \quad sym(g_d(Y),d') \text{ and } D[1] \text{ are in} \\
& \text{different cycles in } g_d(Y) \text{ w.r.t. } D \\
& f\backslash \quad sym(g_d(Y),d!)\wedge D[d!] \\
= \; & c_{o},i,d\text{-},g_d(Y) \text{ if } p(5d(Y),D) = 0 \\
& /\backslash l \;=\; q(g_d(Y),D)>0 \\
& /\backslash \text{sym}(<\text{fo}(Y),l)=D[d']
\end{aligned}
$$

3.3 Number of virtual channels

We now show that the number of virtual channels used in our algorithm of the previous section is sufficient. When the first routing rule is used and routing is done over a cycle-merging channel Cj_{kdY}, $k = p(Y, D)$. An upper bound on $p(Y, D)$ is $\backslash n - dll\backslash$ since the Cycle-Merge algorithm chooses the leftmost symbol d in Y which is in one of the k cycles. There are at most $in - d) + 1$ candidate symbols for selection. Since each of the k cycles requires at least two of these candidates, there can be at most $\backslash n - dll\backslash$ cycles of interest. Hence $k = p(Y, d) < \backslash n - dll\backslash$.

Similarly, when we apply the second rule and route along a cycle-reducing channel c_{OJidY}, $t = ?(Y, D)$ and an obvious upper bound for $q(Y, D)$ is $(« — 1)$.

3.4 Dead-lock free property

Theorem 9: The routing function $T\wedge CMA$ is deadlock-free.

Proof: The following three cases occur while routing using our algorithm.

Case 1. A packet received along a cycle-merging channel is forwarded along a cycle-merging channel

Case 2. A packet received along a cycle-merging channel is forwarded along a cycle-reducing channel

Case 3. A packet received along a cycle-reducing channel is forwarded along a cycle-reducing channel

In Case 2, the channel subscripts are obviously decreasing, since a cycle-merging channel has a 1 in the first position and a cycle-reducing channel has a 0 in the first position. In Case 1, if the incoming channel is $C]_{w_i IY}$ and the outgoing channel is $q\ w',d,g<j,(Y)$ then we claim that $w' = w - 1 < w$. Recall that $w = p(Y, D)$ and $w' = p(gJY), D)$. Since Phase-1 is in progress, each step of Phase-1 reduces the size of yby 1. (See Theorem 6).

In Case 3, if the incoming channel is denoted $c\wedge_{wdY}$ and the outgoing channel is denoted $c_{\varrho}w',d',g_{d},(Y)$, then we claim that $w' = w - 1$. Once again, recall that $w = q(Y, D)$ and $w' = q(gJY), D)$. Phase-2 is being used, and as illustrated in Theorem 6, the function q decreases by 1 after each step of Phase-2.

4 Conclusions

We have presented two deadlock-free routing algorithms for the Star Graph. Both these algorithms are suitable in a Wormhole routing environment. Our algorithms are based on the concept of virtual channels. The first algorithm, called e-star, is similar to the well known e-cube algorithm for hypercubes. Unlike the *e*-cube algorithm, e-star is not deadlock-free, and we achieve deadlock-free property through the introduction of virtual channels. We also presented a new cyde-merging algorithm for routing in an «-star. We showed that the cycle-merging algorithm can be made deadlock-free using virtual channels. The properties of the e-star and Cycle-Merge algorithms are given in Table 3.

Table 3

Routing algo.	Maximum routing dist.	Average routing dist.	Max. no. of virt. chan. per phys. chan.	Avg. no. of virt. chan. per phys. chan.
e-star	*2n-3*	2n+1-3Hn	n−1	7_2
Cycle-Merge	$\left\lfloor \frac{3}{2}(n-1) \right\rfloor$	$n+^2/_n+Wn-4$	$\left\lfloor \frac{3(n-1)}{2} \right\rfloor$	$\left\lfloor \frac{5n}{4} \right\rfloor - 1$

5 References

1 AKERS, S.B., and KRISHNAMURTHY, B.: 'A group-theoretic model for symmetric interconnection networks', *IEEE Trans. Computers*, 1989, 38, pp. 555-566

2 DALLY, W., and SEITZ, C.L.: 'Deadlock-free routing in multiprocessor interconnection networks', *IEEE Trans. Computers*, 1987, C-36, (5)

3 HAYES, J.P., MUDGE, T., and STOUT, Q.F.: 'A microprocessor-based hypercube supercomputer', *IEEE Micro*, October 1986, pp. 6-16

4 HILLIS, W.D.: The connection machine' (MIT Press, 1985)

5 MISIC, J., and JOVANOVIC, Z.: 'Routing function and deadlock avoidance in a star graph interconnection network', /. *Parallel and Distributed Computing*, 1994, pp. 216-228

6 NI, L., and MCKINLEY, P.: 'Tutorial on wormhole routing', *IEEE Computer*, February 1993

7 PASE, D.M., and LARRABEE, A.R.: 'Intel iPSC concurrent computer' in BABB, R.G. (Ed.): 'Programming Parallel Processors', (Chapter 8), (Addison Wesley, Reading, MA, 1987)

8 HAREL, D., AKERS, S.B., and KRISHNAMURTHY, B.: 'The star graph: an attractive alternative to the «-cube', Proc. Int. Conf. Parallel Processing, 1987, pp. 393-400