HYPERCUBE MANAGEMENT IN THE PRESENCE OF NODE FAILURES*

Dilip D. Kandlur and Kang G. Shin

Real-time Computing Laboratory
Department of Electrical Engineering & Computer Science
The University of Michigan
Ann Arbor, MI 48109–2122

Abstract

The problem of allocation and release of subcubes from a hypercube with node failures is addressed. Two algorithms are presented, both based on the Buddy allocation scheme for memory management which is also used by the AXIS operating system of the NCUBE hypercube computer. The first algorithm is a simple variation of the Buddy algorithm which permits the efficient allocation of subcubes in the presence of a single faulty node. The second algorithm, which effectively subsumes the first, tries to reduce the fragmentation caused by multiple failed nodes. It uses a relabeling scheme to group the failed nodes so that large non-faulty subcubes can be detected using the Buddy allocation scheme. The relative performance of these algorithms is studied using simulation and the proposed algorithms are shown to have a consistently better performance. Issues relating to the detection of faulty nodes on the NCUBE computer and the consequences of the relabeling on message passing are also discussed.

1 Introduction

In recent years, hypercube multicomputers have emerged as a very important class of distributed memory multiprocessors. The popularity of this architecture can be attributed to its regular structure and its rich interconnection topology which facilitates the efficient embedding of many useful structures like meshes, trees, and rings of even length [1,2]. It also yields simple and efficient algorithms for node–node communication, broadcasting, scattering, etc. [3], which are important for the development of parallel algorithms. Numerous research efforts related to hypercube architectures, operating systems, programming languages, and scientific computation [4,5,6] have been undertaken, and several research and commercial hypercube multicomputers have been built.

In a multi-tasking environment, it is essential to allocate processors in the hypercube to requesting tasks. The unit of allocation is a subcube with size varying from 0 to D, the size of the hypercube, so that users see their own, albeit smaller, hypercube computers. Since the hypercube nodes are the most important resource in the system, the management of the node processors is an important problem. The subcube allocation problem is similar to the conventional dynamic memory allocation problem in which processes make requests to allocate and release contiguous blocks of memory. An important difference however, is that the connectivity of the hypercube makes it possible for processors to be "contiguous" in many ways. In fact, there are $2^{D-k} \cdot \binom{D}{k}$ distinct (but not disjoint) subcubes of size k in a D-dimensional hypercube, Q_D. However, the procedures available for detection of these possibilities, like the method of prime implicants [7], are in general very complex. In [8], Chen and Shin have proposed strategies based on the Buddy allocation scheme for memory management [9] and on Gray

*This work is supported in part by the Office of Naval Research under contracts N00014–85–K–0122 and N00014–85–K–0531.

codes which try to improve the ability to detect subcubes while maintaining a low time complexity. On the other hand, Dutt and Hayes [10] have proposed a heuristic algorithm which focuses on the coalescing of the cube when a subcube is released. This algorithm however, has an exponential *worst case* complexity.

This paper addresses the problem of subcube allocation when there are some faulty nodes present in the hypercube. Since practical implementation has been an important consideration, only algorithms with a low worst case complexity have been considered. Faults in the hypercube can be classified as link faults and node faults. In this paper, we will deal with node faults in which a node and all its communication links are not available for use. Furthermore, our study is restricted to *static reconfiguration* in which node faults are detected at system initialization and the reconfiguration takes place at that time. The problem of detection of faulty nodes is discussed briefly in Section 4.

In this paper, two processor allocation strategies are presented in Section 2 and a comparison of their performance can be found in Section 3. In Section 4, issues relating to the detection of faulty nodes and the implications of node relabeling on the communication scheme as related to the NCUBE/six hypercube computer are discussed. Conclusions are drawn in Section 5.

2 Allocation Algorithms

The subcube allocation problem, with request sizes in powers of 2, fits in well with the Buddy allocation scheme. The hypercube graph can be represented by a bit vector indicating allocated and free nodes where the position i corresponds to the node labeled i in the hypercube. Whenever a request is made for a subcube of dimension k, the bit vector is searched for a region of contiguous free nodes of length 2^k starting at positions $i.2^k$ where $i = 0, 1, 2, \ldots$. This is the form used in the AXIS operating system. This scheme has been proved to be *statically optimal*, that is, optimal when processor relinquishment is not considered [8]. An allocation strategy is said to be *statically optimal* if it can accommodate any input request sequence $\{I_i\}_{i=1}^{k}$ iff $\sum_{i=1}^{k} 2^{|I_i|} \leq 2^D$ where $|I_i|$ is the dimension of the requested subcube.

```
Allocation Vector with node 4 faulty:

        0 0 0 0 1 0 0 0

1-cube allocated:

        1 1 0 0 1 0 0 0

Now, a 2-cube cannot be allocated

(a) : AXIS allocation strategy


Lists of free subcubes when node 4 is faulty

        list 0:    5
        list 1:    6
        list 2:    0
        list 3:    -

 Request 1-cube:  subcube at 6 allocated
 Request 2-cube:  subcube at 0 allocated

(b) Standard Buddy Allocation strategy
```

Figure 1: A 3-cube with node 4 faulty and request sequence $\{Q_1, Q_2\}$

```
Initial state of the Allocation Vector:

        1 0 0 0 1 0 0 0

The faulty nodes have labels:  0 0 0
                               1 0 0

                   Direction:   3 2 1

which differ in direction 3


Map the faulty directions to the low directions

          dirn:  1 -> 2
                 2 -> 3
                 3 -> 1


The new labels of the faulty nodes are: 0 0 0
                                        0 0 1

The Allocation Vector for the new labelling is:

        1 1 0 0 0 0 0 0
```

Figure 2: A 3-cube with 2 faulty nodes showing fragmentation

This algorithm can handle faulty nodes by treating them as permanently allocated. We will call this modified scheme Algorithm 1 in the rest of the paper. Although the algorithm is statically optimal, in the case of node failures it loses this property. For example in Fig. 1, which shows a Q_3 with node 4 faulty, the allocation request sequence $\{Q_1,Q_2\}$ cannot be granted. This property can be preserved for single node failure by the use of a standard Buddy allocation scheme with separate lists for subcubes of different sizes. The procedures for allocation and release in this scheme are given in Fig. 3. Using this algorithm, as shown in Fig. 1, the request sequence $\{Q_1,Q_2\}$ can be satisfied. The proof of static optimality is easy and follows the lines of the proof for the static optimality given in [8]. The basic steps in the proof are:

Step 1 There can be no more than one free node in any list.

Step 2 Since $\sum_{i=0}^{i<j} 2^i < 2^j$, the total number of nodes in a subcube of size j is greater than the number of nodes in all free subcubes of size less than j.

Step 3 For a valid request, $2^{|\mathbf{I}_r|} \leq$ the number of free nodes and by Steps 1,2 the free nodes cannot all be contained in subcubes of size less that $|\mathbf{I}_r|$. Hence, there exists a free subcube of size $\geq |\mathbf{I}_r|$.

This ensures that every valid request can be honored and so, the algorithm is statically optimal.

When there is more than one node fault, the faulty nodes may be widely dispersed in the bit vector, although they may be neighbors in the hypercube graph. This results in fragmentation of the bit vector with a consequent reduction in the ability to grant requests. This can be seen in the example shown in Fig. 2 in which no Q_2 can be allocated. This fragmentation is in part a result of the poor ability of the Buddy system in recognizing subcubes. It recognizes only those subcubes whose labelings are of the form $b_D b_{D-1} \ldots b_{D-k} *^k$. Thus, fragmentation would be minimized if the faulty nodes differ in the small numbered bit positions since they would be closely packed in the bit vector (Fig. 2). This suggests that a logical relabeling of nodes could be used to reduce the fragmentation which is the central idea used in the

```
Algorithm Allocate (dim);
  /*
     maintains separate lists for subcubes of different sizes
     with list[i] containing the subcubes of dimension i.
     D is the dimension of the hypercube.
  */

  if (list[dim] not EMPTY)
     allocate the subcube from the head of list[dim];
  else
     begin
        search = dim + 1;
        while ( search ≤ D ) and ( list[search] EMPTY )
           search = search + 1;

        if (search > D)
           /* no subcube can be allocated */
           return(FALSE);
        else
           begin
              remove the subcube at the head of the list,
              let b be the start address of this subcube.
              allocate nodes [b, b + 2^dim - 1] to the requestor.
              split = search - 1;
              while ( split ≥ dim )
                 insert a subcube at the head of list[split]
                 with start address (b + 2^split);
           end;
     end;
end; {allocate}


Algorithm Release (dim, cube_addr);
  /*
     tries to form as big a subcube as possible by coalescing
     the released subcube with its buddy if that is available.
  */

  buddy = cube_addr bit_xor 2^dim;
  search list[dim] for a subcube with start address = buddy;

  if (found)
     begin
        remove the buddy subcube from list[dim];
        cube_addr = min(cube_addr,buddy);
        Release( (dim+1), cube_addr);
     end;
  else
     add a subcube with start address = cube_addr
     to the head of list[dim];

end; {release}
```

Figure 3: Buddy Allocation and Release

```
Algorithm Relabel (num_faults, fault_list);
   /* first determine the fault directions */
   dim = 0;
   fault_1 = fault_list[1];
   for i = 2 to num_faults
      dim = dim bit_or (fault_1 bit_xor fault_list[i]);

   if (dim = 0) or (dim = 2^D - 1)
      /* none or all are fault directions: Identity mapping */
      for d = 1 to D
         map[d] = d;
   else
      begin
         /* map the directions */
         num_dirns = number_of_ones(dirn);
            /* number of fault directions */
         good_dirn = num_dirns + 1;
         bad_dirn = 1;
         for d = 1 to D
            if (dim bit_and 2^d - 1) /* is direction d faulty ? */
               begin
                  map[d] = bad_dirn;
                  bad_dirn = bad_dirn + 1;
               end;
            else
               begin
                  map[d] = good_dirn;
                  good_dirn = good_dirn + 1;
               end;
      end;

   /* now map all faulty nodes */
   for i = 0 to 2^D - 1
      faulty[i] = FALSE;

   for i = 1 to num_faults
      faulty[map_fault(fault_list[i],map)] = TRUE;

   /* allocate all nodes and release only the good nodes */
   addr = Allocate(D);
   for j = 0 to 2^D - 1
      if (not faulty[j])
         Release(0,j);

end; {relabel}

Routine map_fault(node, map) does a permutation on
the bits of the node label with the ith
bit transformed to map[i].
```

Figure 4: The Node Relabeling Algorithm

algorithm described in Fig. 4. The main steps in this algorithm which is referred to as Algorithm 2 are:

Step 1 Find the minimal subcube which contains all the faulty nodes. This is done by finding the set of directions in which the labels of the faulty nodes differ.

Step 2 Map these directions to the low order bit positions to obtain a direction transformation. For example, in Fig. 5, where the faulty nodes are numbered 21, 29, and 5, the transformation would be from 654321 to 643215. Apply the transformation to the old labels to obtain new node labels.

Step 3 Allocate all nodes from the hypercube as subcubes of size 0 and release only those which are not faulty. The standard Buddy allocation scheme described in Fig. 3 is used for these and subsequent allocation and release operations. The subcube release algorithm automatically forms lists of subcubes of different sizes.

The faulty nodes then remain permanently allocated until the hypercube is re-initialized. It can be seen that the new labeling is *consistent* and any two nodes with labels differing in only one bit position have an edge between them. This is because the transformation is basically a permutation of the bits of the labeling.

3 Simulation Results

The processor allocation algorithms discussed in the previous section have been simulated on a DEC VAX-11/780 computer. Their relative performance was investigated using two different performance measures. The first measure is R, the percentage of valid incoming requests for which subcubes were allocated. A *valid request* is defined as one where the number of processors requested is less than or equal to the number of free processors. The second measure is U, the average utilization of a processor. This is the percentage of the total time for which a processor was allocated.

In all the experiments, the incoming requests had an inter-arrival time taken from an exponential distribution with a mean of 5 time units. The dimensions of the subcubes requested followed a uniform

distribution between 0 and D, the dimension of the hypercube. The subcube residence time, that is, the length of time an allocated subcube is occupied, followed an exponential distribution with a given mean which varied with the experiment. The experiments were aimed at finding the range of variation in performance between the two algorithms. The measure U is computed as $\sum_{i=1}^{\#grants} 2^{|\mathrm{I}_i|}.t_i/(T \times 2^D)$ where $|\mathrm{I}_i|$ and t_i are respectively the dimension and residence time of the subcube acquired by $|\mathrm{I}_i|$, and T is the total simulation time.

The cases which favor the first algorithm are those in which the faulty nodes are "contiguous" in the original bit vector and form a subcube. In this case, the mapping function is identity and the only difference between the two algorithms is that Algorithm 2 uses a *best-fit* approach in that it maintains separate lists for subcubes of different sizes, while Algorithm 1 uses a *first-fit* approach. In this case, the experiments showed no discernible difference in performance which is consistent with reported observations on the relative performance of *best-fit* and *first-fit* strategies for dynamic memory allocation [11].

The cases which favor Algorithm 2 are those in which the faulty nodes differ in the high order bits, for example, nodes 0 and 32 in a Q_6. Tables 1 and 2 summarize the results of the experiments in this category where the number of faulty nodes is 2 and the size of the hypercube is varied from 5 to 10. (In Tables 1–4, the first figure in each entry is for Algorithm 1 while the second is for Algorithm 2.) Since the arrival rate is held constant, the residence time determines the load on the system. It is observed that as the load increases, U increases and R decreases. Algorithm 2 performs better than Algorithm 1 by 5-15 % on both measures. The average utilization decreases as the size of Q_n increases because the ratio: average number of nodes requested to total number of nodes decreases.

For the average case analysis, the mean residence time was fixed at 20 time units. Faulty nodes were determined using a uniform random number generator and for each Q_n and number of faulty nodes, the experiment was repeated 50 times with different sets of faulty nodes. The performance measures R and U were averaged over these experiments. Tables 3 and 4 give a summary of the results for cases where the number of faulty nodes varies from 1 to 4. Some of the results are also shown in graphical form in Fig. 6 and Fig. 7. When the number of failed nodes was less than four, the performance figures for both algorithms were close to their "extreme case" values. This could be explained for the 2 faulty node case as follows.

Assuming a uniform distribution, the expected separation between two faulty nodes in the bit vector is $2^D/2 = 2^{D-1}$. At the same time, the mean *Hamming distance* between them is $\sum_{d=0}^{D} d \cdot \binom{D}{d \approx D/2}$ since there are $\binom{D}{d}$ nodes at a distance d from a given node. When Algorithm 2 is used, the two faulty nodes would be packed into a subcube of expected size $D/2$ and so, their expected separation would be $2^{D/2}$ which is small compared to 2^{D-1}. Hence, fragmentation is reduced considerably by the use of Algorithm 2.

When the number of faulty nodes is four, the results show large fluctuations but even in this case, Algorithm 2 always performs better. Assuming a constant reliability figure for the nodes, the probability of occurrence of k node faults decreases rapidly as k increases. Hence the cases where the number of node faults is small are of more practical importance. In these cases, it is observed that Algorithm 2 performs consistently better than Algorithm 1.

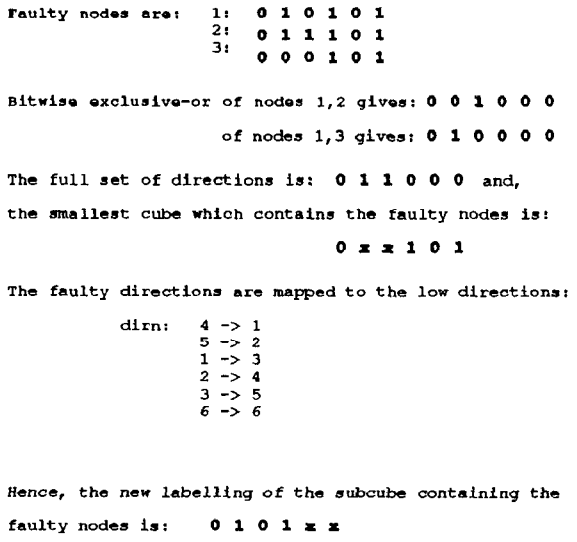

Figure 5: Relabeling of a 6-cube with 3 faulty nodes

Cube Size	Mean Residence Time		
	20	40	80
5	32.27	49.82	64.07
	43.59	60.26	71.14
6	27.87	48.87	65.86
	42.05	57.10	72.05
7	25.91	43.18	64.24
	41.01	56.50	70.05
8	24.02	44.14	59.54
	36.89	53.34	69.69
9	19.30	37.96	57.44
	38.63	52.06	65.49
10	19.98	36.74	58.86
	35.62	49.65	63.87

Table 2: Node Utilization U: "extreme" case for 2 faulty nodes.

Cube Size	Mean Residence Time		
	20	40	80
5	80.03	79.65	74.02
	89.76	85.93	78.71
6	84.46	84.60	81.41
	93.53	90.55	87.14
7	84.88	85.69	84.77
	95.54	92.08	89.56
8	88.26	90.02	88.37
	96.84	95.47	93.03
9	88.80	88.58	89.07
	98.65	95.96	94.21
10	89.72	90.61	90.58
	99.03	96.09	93.65

Table 1: Requests granted R: "extreme" case for 2 faulty nodes.

Cube Size	Number of faulty nodes			
	1	2	3	4
5	84.25	78.94	74.84	68.44
	92.84	88.95	83.10	81.78
6	87.69	83.91	79.65	75.07
	95.44	91.44	87.85	78.32
7	88.99	84.47	81.62	78.62
	96.74	94.61	90.31	83.65
8	90.46	88.30	85.36	83.05
	97.70	95.68	92.79	89.52
9	91.67	89.42	86.69	84.46
	98.59	97.41	94.64	93.29
10	93.15	90.25	88.66	86.76
	99.02	97.82	96.12	91.42

Table 3: Requests granted R: average case.

Cube Size	Number of faulty nodes			
	1	2	3	4
5	37.53	33.04	29.65	23.65
	46.41	43.93	38.90	40.01
6	33.81	30.06	25.93	22.47
	43.88	40.53	35.69	25.31
7	32.53	26.98	23.79	20.67
	41.95	40.51	36.47	27.30
8	27.24	24.93	21.98	19.75
	38.21	36.07	31.73	27.06
9	27.48	24.25	20.89	17.76
	37.51	36.69	33.73	32.70
10	26.54	21.61	19.43	17.38
	35.28	33.89	32.02	25.78

Table 4: Node Utilization U: average case.

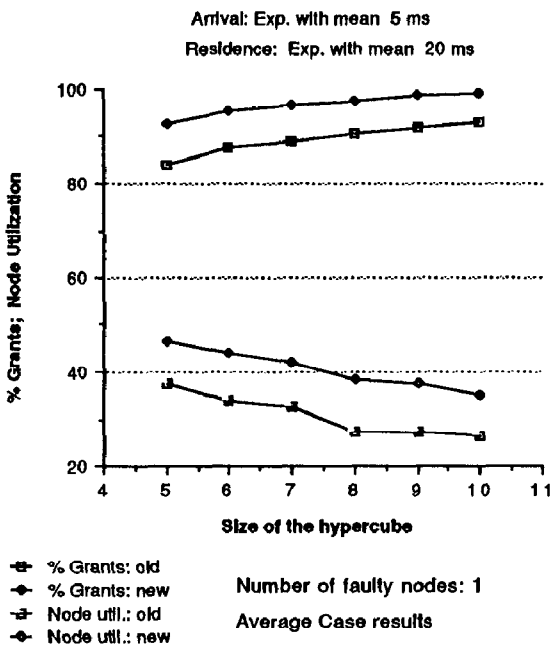

Figure 6: Single Faulty Node

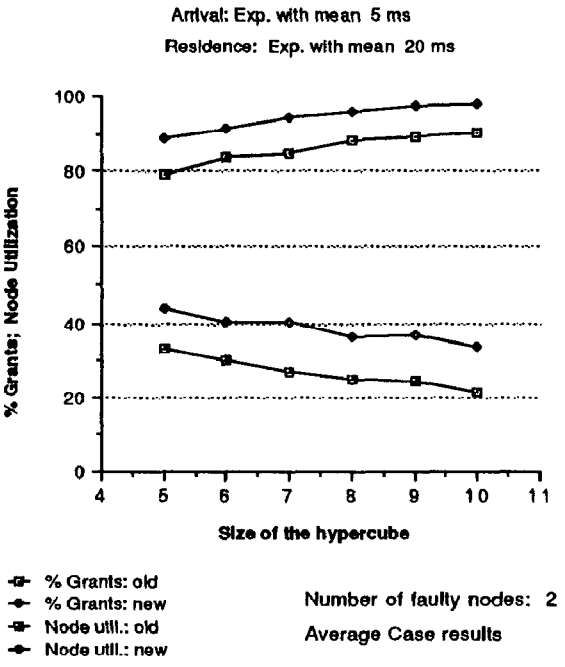

Figure 7: Two Faulty Nodes

4 Practical Considerations

The AXIS operating system which runs on the host processor treats the hypercube as a device, */dev/ncube*, and the subcube allocation and release operations are performed using the *open* and *close* system calls on this device. These operations are executed by the device driver for this device. The management strategy presented in this paper can be implemented by modifying this device driver.

The detection of faulty nodes is linked to the configuration of the NCUBE machine and the interface between the host processor and the hypercube array. In the NCUBE/six system, there are 16 I/O processors on the host processor card which have one link each to a node in the processor array (Fig. 8). On the other hand, the host and I/O processors communicate via shared memory. The system is initialized in stages with the I/O processors initialized in the first stage. In the next stage, processors directly connected to an I/O processor are initialized. The initialization of nodes at a distance of 2 and 3 communication links from the nearest I/O processor takes place in subsequent stages. At the end of each stage, the processors which were initialized in that stage send a message back to the host. Using this mechanism, faulty nodes can be detected as those which fail to report back to the host.

A faulty node can affect the host–node communica-

tions and interfere with the initialization process since its communication links are not available for use. For example, in Fig. 9, the failure of node 0 would affect the initialization of nodes 1, 2 and 3. The host would then have to compute an alternate route to get to these nodes, say using nodes 4, 5. This is essentially the problem of host – node communication in the presence of faulty nodes. In the case of the NCUBE/six, this could probably be solved by examining and evaluating the different possibilities and providing *a priori* alternate routes to handle the situations.

A related problem is that of node–node communication within a subcube. The VERTEX operating system, which runs on the nodes in the hypercube, uses a deterministic routing scheme which associates the logical node numbers with the physical communication channels. For example, a message from $b_D b_{D-1} \ldots b_2 b_1$ to $b_D b_{D-1} \ldots b_2 \bar{b_1}$ would be transmitted on channel 1. This routing scheme would have to be modified to include a mapping from the new logical node numbers to the communication channels. An array giving the map of the old directions to the new would provide the information necessary for the routing.

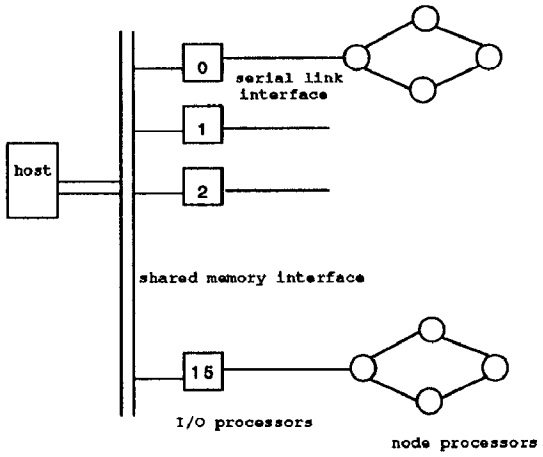

Figure 8: NCUBE/six Configuration

5 Conclusions

An algorithm which tries to reduce the fragmentation caused by node failures by an appropriate relabeling of the nodes has been presented. This algorithm has the same time complexity as the Buddy allocation algorithm which is used by the AXIS operating system on the NCUBE. Simulation results have shown that this algorithm performs significantly better than the NCUBE algorithm when there are very few faulty nodes. To implement this algorithm on the NCUBE, the AXIS operating system and the VERTEX communication kernel would have to be modified. Although the changes required are significant, many of them relate to detection of node faults, a task which is essential to any algorithm handling node faults.

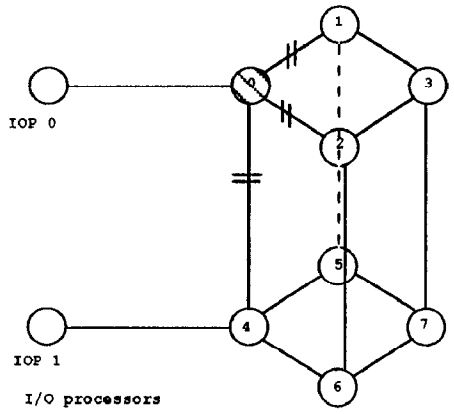

Figure 9: Effects of node failure on communication

References

[1] F. Harary, J. P. Hayes, and H. J. Wu, "A survey of the theory of hypercube graphs", *Computers and Math. with Applications*, 1988, to appear.

[2] Y. Saad and M. Schultz, *Topological Properties of Hypercubes*, Technical Report 389, Department of Computer Science, Yale University, June 1985.

[3] Y. Saad and M. Schultz, *Data Communications in Hypercubes*, Technical Report 428, Department of Computer Science, Yale University, October 1985.

[4] D. Jefferson and B. Beckman, "Virtual Time and Time Warp on the JPL Hypercube", in *SIAM Conference on Hypercube Multiprocessors*, pp. 111–122, 1986.

[5] N. Carriero and D. Gelertner, "Linda on Hypercube Multicomputers", in *SIAM Conference on Hypercube Multiprocessors*, pp. 45–56, 1986.

[6] D. Krumme, K. Venkataraman, and G. Cybenko, "Hypercube Embedding is NP-Complete", in *SIAM Conference on Hypercube Multiprocessors*, pp. 148–160, 1986.

[7] Z. Kohavi, *Switching and Finite Automata Theory*, McGraw Hill, 1978.

[8] M. -S. Chen and K. G. Shin, "Processor Allocation in an n-cube multiprocessor using Gray codes", *IEEE Transactions on Computers*, vol. C-36, no. 12, pp. 1396–1407, December 1987.

[9] D. E. Knuth, *The Art of Computer Programming, Fundamental Algorithms*, Addison Wesley, 1969, pp 442-445.

[10] S. Dutt and J. P. Hayes, "On Allocating Subcubes in a Hypercube Multiprocessor", in *Third Conference on Hypercube Computers and Applications*, Pasadena, California, January 1988.

[11] J. E. Shore, "On the External Storage Fragmentation produced by First-fit and Best-fit Allocation Strategies", *Communications of the ACM*, vol. 18, no. 8, pp. 433–440, 1975.