90-TN

ISMM

INTELLIGENT DISTRIBUTED PROCESSING

Ft. Lauderdale, Florida
December 13-15, 1989

EDITOR: R. Ammar

A Publication of
The International Society for
Mini and Microcomputers - ISMM

ISBN 0-88986-138-2

ACTA PRESS
ANAHEIM * CALGARY * ZURICH

ISMM

Proceedings of the ISMM International Conference, Intelligent Distributed Processing, held in Fort Lauderdale, Florida, U.S.A. December 13-15, 1989.

SPONSOR
The International Society for Mini and Microcomputers — ISMM
Technical Committee on Computers

Editor: R. Ammar

ACTA PRESS
P.O. Box 2481
Anaheim, CA
U.S.A. 92814

ACTA PRESS
P.O. Box 3243, Stn. "B"
Calgary, Alberta
Canada T2M 4L8

ACTA PRESS
P.O. Box 354
CH-8053 Zurich
Switzerland

ACTA PRESS CODE: 152

A Three-Node Grouping Problem in the Hypercube Networks

Qiang Li, Ling Wang and Naphtali Rishe

School of Computer Science
Florida International University
The State University of Florida at Miami
Miami, FL 33199

ABSTRACT

This paper presents a solution to the following problem: Given two nodes in a binary hypercube, a third node that satisfies certain conditions is to be selected to cooperate with them in solving a problem. The selection strategy of the third node has direct impact on the efficiency of the system involved, especially on the system traffic load. In this paper, a solution to the problem and an algorithm which selects the third node are presented.
Keyword: Hypercube, Shortest distance, Traffic reduction, Three-node grouping.

1 Definition of the problem

A binary hypercube can be defined as follows: An N-dimensional binary hypercube has 2^N processors, each with a unique binary number, from 0 to $2^N - 1$, as its id number. Two nodes are connected iff their id numbers differ in exactly one bit position. The connection between two nodes consists of two unidirectional channels, one in each direction.

Let $d(n_1, n_2)$ be the distance between nodes n_1 and n_2. It has been shown that the shortest distance between two nodes is equal to the number of bit positions in which their id numbers disagree. We define $d(n_1, n_2, n)$ as the shortest distance from n_1 to n_2 through n (i.e. $d(n_1, n_2, n) = d(n_1, n) + d(n, n_2)$.) Let $f(n_1, n_2, n)$ be some boolean function which yields true true value if the three arguments meet certain conditions. The three-node grouping problem can be defined as follows.

Definition:

Given two nodes in a hypercube, n_1 and n_2, for all the nodes n in the hypercube such that $f(n_1, n_2, n)$ is true: (1) find the node n which has minimum $d(n_1, n_2, n)$ and (2) if several n's yields the minimum $d(n_1, n_2, n)$, find the one with minimum $d(n_1, n)$ (or minimum $d(n_2, n)$.) #

The problem is raised in a system in which a task is performed by three cooperating nodes in a hypercube; two of them are given by the prior tasks and the third can be selected anywhere in the hypercube. Minimizing distance from n_1 to n_2 through n can reduce the communication traffic in the system and the speed of performing the task. The reason for the second part of the definition is that minimizing the distance between n_1 and n (or between n_2 and n) can help to reduce the chance of encountering inconsistent data status when many tasks are carried out in parallel in the same system. Since we can only minimize the distance between n and one of n_1 and n_2, we try to minimize $d(n_1, n)$ in the following discussion, and minimizing $d(n_2, n)$ is symmetric.

An implication of the above definition is that the processor which makes the selection needs the knowledge about all the three processors.

The problem was raised originally in a multiprocessor parallel architecture CONNECT [2]. CONNECT is the hardware foundation for a massively parallel database machine LSDM [3]. In CONNECT, a group of processors are used to control a *three-stage Clos network* [1]. The processors of the group are linked into a hypercube network. The cooperation of three processors are needed to set up a circuit in the Clos network; n_1 and n_2 are given by the static network connections and n is to be selected according to the current status of the Clos network. By using the selection strategy described in the following sections, the data traffic in the hypercube network is reduced by 20% comparing to the uncontrolled selection strategies.

2 The Selection Strategy

Let H be a hypercube of dimension N, and n_1 and n_2 are two nodes of H such that they are different in M bit positions, where $0 \leq M \leq N$. Let $insidebit_{n_1,n_2}$ denote the set of bit positions where n_1 and n_2 do not agree, and $outsidebit_{n_1,n_2}$ denote the set of bit positions where n_1 and n_2 agree. By definition, the cardinality of the set $insidebit_{n_1,n_2}$ is M and the cardinality of $outsidebit_{n_1,n_2}$ is $N - M$. For example, if n_1 = 00100100 and n_2 = 00001111, then $M = 4$, $insidebit_{n_1,n_2}$ = {0, 1, 3, 5} and $outsidebit_{n_1,n_2}$ = {2, 4, 6, 7}.

We say that $H_s(n_1, n_2)$ is a sub-hypercube of H defined by n_1

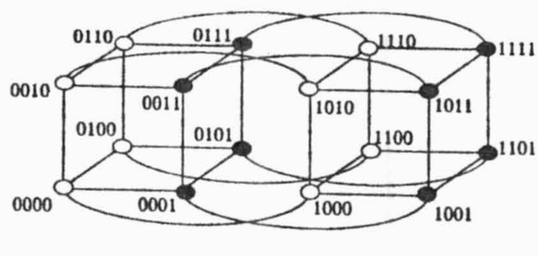

Figure 1: A sub-hypercube

This research has been supported in part by a grant from Florida High Technology and Industry Council

and n_2 if H_s consists of all the nodes of H such that their id numbers agree with n_1 and n_2 at the bit positions in $outsidebit_{n_1,n_2}$. In other words, $H_s(n_1, n_2)$ can be generated by keeping the bit positions in $outsidebit_{n_1,n_2}$ the same as in n_1 or n_2 and listing all the combinations of the M bit positions in $insidebit_{n_1,n_2}$. It is not difficult to verify that, according to the definition of a binary hypercube, $H_s(n_1, n_2)$ is indeed an M-dimensional binary hypercube. Figure 1 shows a 3-dimensional subhypercube defined by node 0001 and 1111. The solid nodes are inside $H_s(0001, 1111)$.

Lemma

Let H be an N-dimensional hypercube and $H_s(n_1, n_2)$ be an M-dimensional sub-hypercube of H. Let n be any node of H and let t be the number of bit positions in $outsidebit_{n_1,n_2}$ such that n disagrees with n_1 or n_2 at those positions. Then, $d(n_1, n_2, n) = M + 2 * t$.

Proof:

Assume that n_1 and n do not agree at x bit positions and x_{in} of them are in $insidebit_{n_1,n_2}$ and x_{out} of them are in $outsidebit_{n_1,n_2}$. Also, assume that n and n_2 do not agree at y bit positions and y_{in} of them are in $insidebit_{n_1,n_2}$ and y_{out} of them are in $outsidebit_{n_1,n_2}$. Then

$$d(n_1, n_2, n) = x + y = x_{in} + x_{out} + y_{in} + y_{out}.$$

Since n_1 and n_2 agree at all bit positions in $outsidebit_{n_1,n_2}$, it must be that $x_{out} = y_{out} = t$. Further, since n_1 and n_2 disagree at all the M bit positions in $insidebit_{n_1,n_2}$, if n agrees with one of them at a bit position in $insidebit_{n_1,n_2}$, it must disagree with the other at the same bit position. Thus, since n agrees with n_1 at $M - x_{in}$ bit positions in $insidebit_{n_1,n_2}$, it must be that $y_{in} = M - x_{in}$. Therefore

$$d(n_1, n_2, n) = x_{in} + x_{out} + y_{in} + y_{out} = M + 2 * t$$

#

Selection Strategy:
According to the Lemma, in order to minimize $d(n_1, n_2, n)$, we should select n so that $t = 0$, i.e., a node n in $H_s(n_1, n_2)$ should be selected for which $f(n_1, n_2, n)$ is true. If no such node is available, a node with the minimum t should be selected. If more than one node have the same minimum d value, the one with least different bits with n_1 should be selected.

3 The Selection Algorithm

The algorithm we need is basically to check the value of $f(n_1, n_2, n)$ according to a given sequence of n's. The node numbers in the sequence are ordered according to the Strategy described above, so the first n which causes $f(n_1, n_2, n)$ to be true will be selected. The following is a description of the algorithm.

```
initialize(n1, n2);
loop
    n := NextNode
    exit when f(n1, n2, n);
end loop;
```

The function *NextNode* is defined as follows: After the initialization, the calls to the function *NextNode* generate a sequence of node numbers in the order specified by the Strategy.

In general, the sequence can be generated by manipulating bit positions in the id numbers. But the algorithms normally involve variable number of nested loops or stack manipulations, and are very difficult to be embedded into other algorithms. Generating the entire sequence and then checking the f function value is not efficient.

In the followings, we present an algorithm which satisfies the requirements on the function *NextNode*. The technique used is similar to the shift register method. The algorithm is presented in an ADA form. To save space, many variable declarations, routine definitions and related packages are omitted.

Given **node1** and **node2**, once the algorithm is initialized, every time the function **next_node** is called, next node number in the sequence defined in the previous section is returned, except that the inside node numbers are generated in the order such that the nodes with shortest distance to **node1** are generated first.

```
package body sequence is

  type node_type is record
                        node_num          : bit_string;
                        last_bit_flipped  : integer;
                      end record;

  type table_type is  array(-1..max_num_nodes) of
node_type;

  table               : table_type;

  current_template     : integer;
  next_avail_slot      : integer;
  num_of_inside_bits   : integer;
  num_of_outside_bits  : integer;
  last_bit_to_flip     : integer;


PROCEDURE initial( node1, node2 : bit_string) IS
  BEGIN
    n := node1 XOR node2;
    num_of_inside_bits := (num of 1-bit in n);
    map(1..num_of_inside_bits)
                      := (bit positions of 1-bit in n);
    map(num_of_inside_bits+1 .. total_num_of_bits)
                      := (bit positions of 0-bit in n);
    last_bit_to_flip := num_of_inside_bits;

    num_of_nodes_to_be_generated
                      := 2 ** num_of_inside_bits - 1;

    -- table(-1) is just to get algorithm started
    table(-1).last_bit_flipped := last_bit_to_flip;
    table(0).last_bit_flipped := 0;
    table(0).node_num := node1;

    current_template := -1;
    next_avail_slot := 1;

  END;

  FUNCTION next_node RETURN bit_string IS
    bit_pos   : integer;
    node_num  : bit_string;
  BEGIN

    IF next_avail_slot > num_of_nodes_to_be_generated
  THEN
      -- the node id inside the subhypercube have been
exhausted
      -- reset parameters to generate the outside nodes.
      current_template := -1;
      last_bit_to_flip := total_num_of_bits;
      table(-1).last_bit_flipped := last_bit_to_flip;
```

```
    num_of_nodes_to_be_generated := 2 **
total_num_of_bits - 1;
  END IF;

  LOOP
    EXIT WHEN table(current_template).last_bit_flipped
              < last_bit_to_flip;
    current_template := current_template + 1;
  END LOOP;

  node_num := table(current_template).node_num;
  bit_pos
    := table(current_template).last_bit_flipped + 1;
  table(current_template).last_bit_flipped := bit_pos;
  node_num(map(bit_pos))
       := node_num(map(bit_pos)) xor 1;
  table(next_avail_slot).node_num := node_num;
  table(next_avail_slot).last_bit_flipped := bit_pos;

  next_avail_slot := next_avail_slot + 1;

  RETURN(node_num);

END;

END sequence;
```

The general idea is that starting from **node1**, all the generated numbers are used as templates to generate the numbers following them. Although it is simple, the algorithm is a little bit hard to explain. The best way to understand it is probably to study an example which is given in the following.

Example:

The following is assumed:

The binary numbers are 4 bits long.
node1 is 0000.
node2 is 0011.

Under the assumption, there are 2 inside bits and 2 outside bits. The two numbers are so chosen that it is easier to show the algorithm. In the case where their different bits are not the lowest bits, the appropriate positions will be obtained by the array **map** which is initialized at beginning of the algorithm.

The following pictures are the content changes of the variable **table** in the algorithm after each call to the function *NextNode*. The upper row is the node numbers and the lower row is the corresponding **last_bit_flipped**. "T" indicates the current template and "N" indicates the next available slot.

Each template is used to generate the subsequenct numbers until all its bit positions less or equal to **last_bit_to_flip** are flipped. The flipping starts from the lower bit position to higher bit position (from right to left). At each step, the number generated (the last one in the upper row) is obtained from the number pointed to by "T" by flipping the bit position which is in the same box of "T". Notice that step 4 is the only time when the position of the current template goes back towards left since step 3 generates the last inside node number. Also, the last inside node number is always **node2**.

4 Conclusion

The selection strategy and the algorithm presented can be used in any application which has similar requirements. The application of the algorithm in the architecture CONNECT has reduced the system communication traffic significantly.

Initial step:

???	0000													...
T 2	0	N												

After step 1:

???	0000	0001												...
2	T 1	1	N											

After step 2:

???	0000	0001	0010											...
2	T 2	1	2	N										

After step 3:

???	0000	0001	0010	0011										...
2	2	T 2	2	2	N									

After step 4:

???	0000	0001	0010	0011	0100									...
4	T 3	2	2	2	3	N								

After step 5:

???	0000	0001	0010	0011	0100	1000								...
4	T 4	2	2	2	3	4	N							

After step 6:

???	0000	0001	0010	0011	0100	1000	0101							...
4	4	T 3	2	2	3	4	3	N						

After step 7:

???	0000	0001	0010	0011	0100	1000	0101	1001						...
4	4	T 4	2	2	3	4	3	4	N					

After step 8:

???	0000	0001	0010	0011	0100	1000	0101	1001	0110					...
4	4	4	T 3	2	3	4	3	4	3	N				

After step 9:

???	0000	0001	0010	0011	0100	1000	0101	1001	0110	1010				...
4	4	4	T 4	2	3	4	3	4	3	4	N			

After step 10:

???	0000	0001	0010	0011	0100	1000	0101	1001	0110	1010	0111			...
4	4	4	4	T 3	3	4	3	4	3	4	3	N		

After step 11:

???	0000	0001	0010	0011	0100	1000	0101	1001	0110	1010	0111	1011		...
4	4	4	4	T 4	3	4	3	4	3	4	3	4	N	

After step 12:

???	0000	0001	0010	0011	0100	1000	0101	1001	0110	1010	0111	1011	1100	...
4	4	4	4	4	T 4	4	3	4	3	4	3	4	4	N

Figure 2. Data changes in the example

Acknowledgement

The authors gratefully acknowledge the advice of David Barton, Doron Tal, Nagarajan Prabhakaran and Scott Graham.

References

[1] C. Clos. A study of nonblocking switching networks. *The Bell System Technical Journal*, 406–424, Mar 1953.

[2] Qiang Li. *The Architecture and The Related Control Problems of a Transputer Based Highly Parallel Database Computer*. PhD thesis, Florida International University, Miami, FL 33199, August 1989.

[3] N. Rishe, D. Tal, and Q. Li. Architecture for a massively parallel database machine. *Microprocessing and Microprogramming. The Euromicro Journal*, 25:33-38, 1988.