하이퍼큐브 멀티컴퓨터를 위한 효율적인 상호 배제 알고리즘

배 인 한†

요 약

우리는 하이퍼큐브 구조를 위한 효율적인 분산 대칭 상호 배제 알고리즘을 제안한다. 제안된 알고리즘의 기본 개념은 하이퍼큐브에 메쉬를 논리적으로 삽입하는 기법에 기초를 두고 있다. 그리고 제안하는 알고리즘의 성능을 Gupta의 알고리즘과 비교하였다. 성능 비교 결과 최소 왕복 지연은 Gupta의 알고리즘과 같고, 평균 블락킹 지연은 Gupta의 알고리즘 보다 약간 길고, 그리고 임계 자원 액세스당 필요한 제어 메세지 갯수는 Gupta의 알고리즘 보다 작음을 알 수 있었다.

An Efficient Mutual Exclusion Algorithm for Hypercube Multicomputers

Ihn Han Bae†

ABSTRACT

We present an efficient decentralized, symmetric mutual exclusion algorithm for the hypercube architecture. The algorithm is based on the technique which embeds a mesh into a hypercube. We compare the performance of our algorithm with that of Gupta et al.'s algorithm. As a result of performance comparison, the minimum round-trip delay is equal to that of Gupta et al.'s algorithm, the average blocking delay is a little longer than that of Gupta et al.'s algorithm, and the number of messages per access to critical resource is fewer than that of Gupta et al.'s algorithm.

1. Introduction

In the recent years, numerous topologies have been proposed for interconnecting nodes in a multiprocessor system. Among them, the hypercube architecture has earned wide acceptance in both SIMD and MIMD configurations because of its simple, yet rich topology. Efficient solutions for synchronization problems are critical to the performance of many algorithms on MIMD machines. In this paper, we address the problem of mutual exclusion on a hypercube.

The mutual exclusion problem is a well-known and fundamental problem in operating systems. The problem is to guarantee mutually exclusive access to a critical section among a set of competing independent

※This work are supported by Catholic University of Taegu-Hyosung grant, 1996.

† 정 회 원:대구효성가톨릭대학교 전자정보공학부 부교수

논문접수:1995년 10월 30일, 심사완료:1996년 5월 30일

processes. There are several solutions to this problem in distributed systems. One of the first distributed mutual exclusion algorithms was presented in [3]. Ricart and Agrawala improved Lamport's solution by employing deferred grants [6]. Subsequently, Maekawa optimized the number of messages required by Ricart and Agrawala's algorithm to achieve mutual exclusion [4]. Recently, Gupta et al. proposed a decentralized, symmetric mutual exclusion algorithm, called Selective Broadcast, that makes full use of the hypercube architecture [1].

In this paper, we present an efficient decentralized, symmetric mutual exclusion algorithm for the hypercube architecture. We compare the performance of our algorithm with that of Gupta et al.'s mutual exclusion algorithm with respect to the minimum round-trip dealy, the average blocking delay, and the number of messages per access to critical resource.

The remainder of this paper is organized as follows : Section 2 describes the notation and terminology of the hypercube architecture and introduces Gupta et al.'s mutual exclusion algorithm. Section 3 provides an outline of our mutual exclusion algorithm for the hypercube architecture. In Section 4, we calculate the minimum round-trip delay, the average blocking delay, and the number of messages per access to the critical resource. Finally, in section 5 we provide concluding remarks.

2. Preliminaries and Previous Work

2.1 Preliminaries

We employ the notation and definitions presented in [5].

Definition : An n-dimensional hypercube, Q_n, is defined recursively as follows :

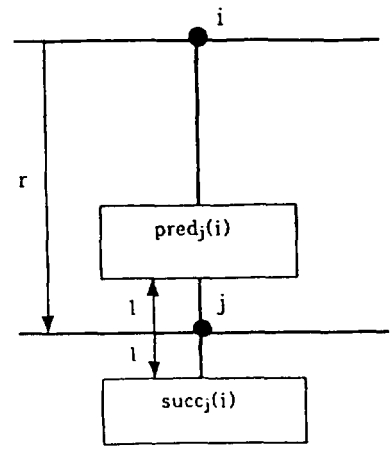
(1) Q_0 is trivial graph with a single node, and

(2) $Q_n = K_2 \times Q_{n-1}$, where K_2 is a complete graph with two nodes, and $\times$ is the product operation on graphs.

An n-dimensional hypercube consists of 2^n nodes. Each node is connected to one another node along each dimension. We number dimensions as d_n, ..., d_2, d_1. The address of a node in Q_n is uniquely represented by an n-bit binary number $(b_n b_{n-1} \ldots b_1)$ where the i-th bit, b_i, corresponds to the coordinate of the node along dimension d_i.

The number of bits differing between the addresses of two nodes provides the distance between these nodes. It is straightforward to verify that two neighboring nodes differ in their addresses in exactly one bit and that the bit correponds to the dimension of the edge joining them. Furthermore, the maximum distance between any two nodes is n, the total number of bits in an address. By inverting all n bits in a address of a node, one obtains the address of the node at distance n from it. Subcubes of an n-cube system are denoted by ternary strings in {0, 1, *}, where * is the Don't Care bit which can be either 0's or 1's.

Each node j has $2 * 2^n$ sets associated with it : $pred_j(i)$ and $succ_j(i)$, $i \in \{1, 2, \ldots, 2^n\}$. Let the distance between nodes i and j be r. $pred_j(i)$ is the set of those neighbors of j that are closer to i by a distance 1 (i.e., a distance $r-1$ from i). In contrast, $succ_j(i)$ is the set of those neighbors of j that are further away from i by a distance 1 (i.e., a distance $r+1$ from i). Note that the cardinalities of $pred_j(i)$ and $succ_j(i)$ sets are 1



(Fig. 1) $Pred_j(i)$ and $succ_j(i)$ sets

$|pred_j(i)| = r$ and $|succ_j(i)| = n - r$. For all i, $pred_i(i) = \phi$ and $succ_i(i)$ contains all n neighbors of node i.

Consider two nodes 00000 and 11000 in Q_5. The distance between 00000 and 11000 is 2. In nodes 00000 and 11000, bit b_5 and b_4 are dissimilar bits, and bit b_3, b_2, and b_1 are similar bits. Node 11000 is joined to 01000 and 10000 along d_5 and d_4, and to nodes 11100, 11010, and 11001 along d_3, d_2, and d_1, respectively. Hence, $pred_{11000}(00000) = \{01000, 10000\}$ and $succ_{11000}(00000) = \{11100, 11010, 11001\}$. Finally, $|pred_{11000}(00000)| = 2$, and $|succ_{11000}(00000)| = 5 - 2 = 3$.

2.2 Previous Work

Gupta et al. proposed a decentralized, symmetric mutual exclusion algorithm, called Selective Broadcast, for the hypercube architecture [1] that is described below with three parts:

Part 1: [What the initiating node does]

Node i broadcasts a REQUEST message to only those $succ_i(i)$ nodes that are reachable along the dimensions d_n, d_{n-1}, ..., d_m, where $m = [\frac{n}{2}]$.

Part 2: [What the intermediate nodes do]

When an intermediate node j at distance r from node i, $1 \le r < m$, receives its (only) REQUEST message along dimension, say d_{min}, it forwards the message to only those $succ_j(i)$ nodes that are reachable along dimensions d_{min-1}, d_{min-2}, ..., d_{m-r}.

Part 3: [When does the algorithm terminate]

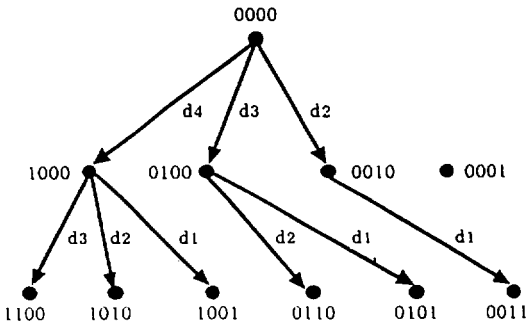
The forwarding of messages terminates at nodes that are a distance m from node i.

Let S_i denote the set of all nodes within a distance of $[\frac{n}{2}]$ of node i, and S_i^* denote the set of nodes reached by the Selective Broadcast. The cardinality of the set S_i^*, denoted by $|S_i^*|$, is given by $|S_i^*| = \left\lceil \frac{n+1}{[\frac{n}{2}]} \right\rceil$. Similarly, the superscript * is attached to the successor and predecessor sets that are actually used in the Selective Broadcast algorithm.

The Selective Broadcast startegy provides a unique way of traveling from node i to any recipient node j in S_i^*. A REQUEST message to node j travels only along the dimensions associated with the dissimilar bits between the addresses of i and j, and it starts along the highest numbered dimension and then successively traverses the lower numbered dimensions. Once a node k at distance $[\frac{n}{2}]$ receives a REQUEST message, it decides whether it can grant the resource to node i. If it can, it sends a GRANT message to its $pred_k^*(i)$ node. An intermediate node k waits until it receives a GRANT message from every node in its $succ_k^*(i)$ set, and if it can grant the resource to node i, then it will send a GRANT message to its $pred_k^*(i)$ node. This backward propagation is performed at all intermediate nodes in S_i^*. In this way, GRANT messages backtrack the path of the REQUEST messages. Once node i receives a GRANT message from every node in its $succ_i^*(i)$ set, it knows that it has the necessary permission from all the nodes in its subset S_i^* and proceeds to access the critical resource. When node i no longer needs the resource, it broadcasts a RELEASE message to every node in S_i^*. The RELEASE messages follow the same paths as the REQUEST messages.

Figure 2 presents an example of a Selective Broadcast when node 0000 in a 4-dimensional hypercube Q_4 wants to access the common resource. We assume that it is the only node vying for the resource. Node 0000 starts the broadcast by sending a REQUEST message to each of the three nodes {1000, 0100, 0010} along dimensions d_4, d_3, and d_2, respectively. Because node 1000 receives the REQUEST message along d_4, it propagates the REQUEST message along d_3, d_2, and d_1 to nodes 1100, 1010, and 1001, respectively. Node 0100 forwards a REQUEST message to nodes 0110 and 0101 along dimensions d_2 and d_1 and node 0010 sends the REQUEST message to node 0011 along the only permissible dimension d_1. This completes the forwarding process.

GRANT messages backtrack the paths followed by REQUEST messages. Since a node j receives a REQUEST message from only one node, it sends only one GRANT message to that same node.



(Fig. 2) The example of the REQUESTs generated in the Selective Broadcast [1].

The performance of Selective Broadcast algorithm is as follows [1];

· Minimum Round-trip Delay

$$D=2\lfloor\frac{n}{2}\rfloor.$$

· Average Blocking Delay

$$B=\frac{1}{N-1}\left[\sum_{d=1}^{\lfloor\frac{n}{2}\rfloor} d\begin{pmatrix}\lfloor\frac{n}{2}\rfloor+d\\ d\end{pmatrix}+2\lfloor\frac{n}{2}\rfloor\right.$$

$$\left.(N-\begin{pmatrix}n+1\\ \lfloor\frac{n}{2}\rfloor\end{pmatrix})\right],$$

where N is the number of nodes in the n-dimensional hypercube.

· Number of Messages per Access to Critical Resource

①in the minimum case (no contention)

$$NM_i=\sum_{d=1}^{\lfloor\frac{n}{2}\rfloor}(3d*\begin{pmatrix}\lfloor\frac{n}{2}\rfloor+d\\ d\end{pmatrix})$$

②in the maximum case (contention at each node in S_i^*)

$$NM_i=\sum_{d=1}^{\lfloor\frac{n}{2}\rfloor}((3\lfloor\frac{n}{2}\rfloor+d+3)*\begin{pmatrix}\lfloor\frac{n}{2}\rfloor+d\\ d\end{pmatrix}),$$

where NM_i denotes the number of messages generated while node i access to one resource.

3. Proposed Mutual Exclusion Algorithm

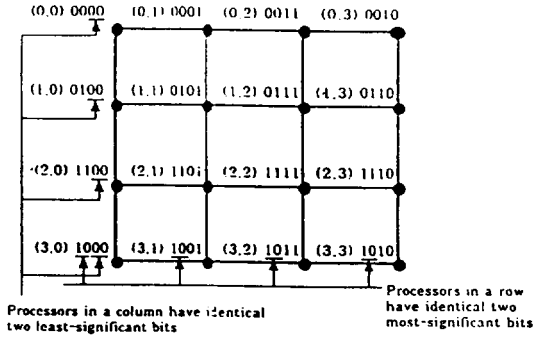
The proposed mutual exclusion algorithm uses the technique of embedding a mesh into a hypercube. We first explain how a mesh is embedded in a hypercube and then explain our mutual exclusion algorithm.

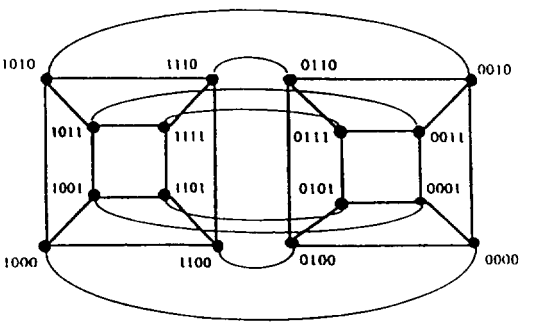
3.1 Embedding a Mesh into a Hypercube

Embedding a mesh into a hypercube is a natural extension of embedding a ring into a hypercube. We can embed a $2^r\times 2^s$ wraparound mesh into a 2^{r+s}-processor hypercube by mapping processor (i, j) of the mesh onto processor G(i, r)‖G(j,s) of the hypercube, where ‖ denotes concatenation of the two Gray codes. Note that immediate neighbors in the mesh are mapped to hypercube processors whose processor labels differ in exactly one bit position. Therefore, this mapping has a dilation of one and a congestion of one [2].

For example, consider embedding a 4×4 mesh into an sixteen-processor hypercube. The values of both r and s are two. Processor (i, j) of the mesh is mapped to processor G(i, 2)‖G(j, 2) of the hypercube. Therefore, processor (0, 0) of the mesh is mapped to processor (0000) of the hypercube. Similarly, processor (0, 1) of the mesh is mapped to processor (0001) of the hypercube, and so on. Figure 3 illustrates embedding of a 4×4 mesh into a 4-dimensional hypercube.

This mapping of a mesh into a hypercube has the following useful properties [2]. All processors in the same row of the mesh are mapped to hypercube processors whose labels have identical r most significant





(Fig. 3) Mapping a 4×4 mesh to processors in a four-dimensional hypercube.

bits. Fixing any r bits in the processor label of an (r + s)-dimensional hypercube yields a subcube of dimension s with 2^s processors. Since each mesh processor is mapped onto a unique processor in the hypercube, and each row in the mesh has 2^s processors, every row in the mesh is mapped to a distinct subcube in the hypercube. Similarly, each column in the mesh is mapped to a distinct subcube in the hypercube.

3.2 The Mutual Exclusion Algorithm

In the mutual exclusion algorithm, the node which requests a critical resource broadcasts REQUEST messages to all the nodes in the same row and the same column as the requesting node in the mesh. In the 4×4 mesh of Figure 3, one set of nodes in the same row as node $(a_4a_3a_2a_1)$ is represented $\{a_4a_3{*}{*}\}$, the other set of nodes in the same column as node $(a_4a_3a_2a_1)$ is represented $\{{*}{*}a_2a_1\}$. If a node $(a_4a_3a_2a_1)$ in the 4-dimensional hypercube requests a critical resource, it broadcasts REQUEST messages to those nodes in the sets of $\{a_4a_3{*}{*}\}$ and $\{{*}{*}a_2a_1\}$. Therefore, the request set of the node $(a_4a_3a_2a_1)$, $S(a_4a_3a_2a_1)$, is defined $\{a_4a_3{*}{*}, {*}{*}a_2a_1\}$. For example, suppose two nodes (1111) and (0000) want to access the critical resource and, consequently, the node (1111) broadcasts REQUEST messages to those nodes in the set {11**, **11} (i.e., {(1100), (1101), (1110), (1111), (0011), (0111), (1011)}), and the node (0000) broadcasts REQUEST messages to those nodes in the set {00**, **00} (i.e., {(0000), (0001), (0010), (0011), (0100), (1000), (1100)}). There exist two common nodes (1100) and (0011) between {11**, **11} and {00**, **00}. If we define our scheme formally, in the n-dimensional hypercube, Q_n, the node $(a_na_{n-1}\ldots a_2a_1)$ which requests critical resource broadcasts REQUEST messages to all the nodes in the request set $S(a_na_{n-1}\ldots a_2a_1)$, where $S(a_na_{n-1}\ldots a_2a_1)=\{a_n\ldots a_{\lfloor\frac{n}{2}\rfloor+1}{*}\ldots{*}, {*}\ldots{*}a_{\lfloor\frac{n}{2}\rfloor}\ldots a_1\}$. The cardinality of the set $S(a_na_{n-1}\ldots a_2a_1)$, $|S(a_na_{n-1}\ldots a_2a_1)| = 2^{\lfloor\frac{n}{2}\rfloor+1}-1$.

Similarly to Gupta et al. 's algorithm, we reproduce the skeleton of the algorithm in [7]. Each node $(a_na_{n-1}\ldots a_2a_1)$ in Q_n executes the following abstract protocol for accessing and releasing the resource:

- REQUEST: Broadcast a timestamped REQUEST message to every node in $S(a_na_{n-1}\ldots a_2a_1)$. Wait for a GRANT message from every node in $S(a_na_{n-1}\ldots a_2a_1)$.
- ACCESS: Access the critical resource.
- RELEASE: Broadcast a RELEASE message to every node in $S(a_na_{n-1}\ldots a_2a_1)$.

The modified algorithm [7] to recover from deadlocks is that the process will know that the system has entered an unsafe state if, after sending a GRANT to a process, it recevies a request from a higher priority process. When this happens, steps can be taken to revoke the GRANT and recover from a deadlock if it has occurred. The modified algorithm introduces three new message types, FAIL, INQUIRE, and YIELD to recover from deadlock situations. A process waiting

for GRANT messages may now cancel a GRANT that has been received and wait for another one, before entering the critical section.

Both node ($a_n a_{n-1} \ldots a_2 a_1$) and node ($b_n b_{n-1} \ldots b_2 b_1$) cannot be granted their request simultaneously, because of the following two observations:

- Since each request set contains all the nodes in the same row and the same column of the requesting node in the mesh system, there will be at least two common nodes between $S(a_n a_{n-1} \ldots a_2 a_1)$ and $S(b_n b_{n-1} \ldots b_2 b_1)$.
- Since a node can grant permission to only one node at a time, any common node can act as an arbitrator and not send grants to both node ($a_n a_{n-1} \ldots a_2 a_1$) and node ($b_n b_{n-1} \ldots b_2 b_1$).

4. Performance Comparison

In this section, we present three popular performance metrics - minimum round-trip delay, average blocking delay, and the number of messages per access and employ them to compare our algorithm with Gupta et al.'s algorithm. We could find that the minimum round-trip delay is equal to that of Gupta et al.'s algorithm, the average blocking delay is a little longer than that of Gupta et al.'s algorithm, and the number of messages per access to critical resource is fewer than that of Gupta et al.'s algorithm.

4.1 Minimum Round-Trip Delay

The minimum round-trip delay for node i, D_i, measures the minimum waiting time of node i from the moment it has requested the resource and before it can be granted the access. Of course, D_i is minimized when there is no contention for the resource. It is called the round-trip delay because it measures the time it takes for REQUESTs to reach all nodes in the request set S_i^* and for all GRANTs to return to node i.

We define the minimum round-trip delay for an algorithm, D, to be the maximum of all D_i's. We define d_{max} to refer to the distance between i and the farthest node in S_i^*. We know that d_{max} is $[\frac{n}{2}]$ for our algorithm because $S(a_n a_{n-1} \ldots a_2 a_1)$ is $\{(a_n \ldots a_{\lfloor \frac{n}{2} \rfloor + 1} * \ldots *), (* \ldots * a_{\lfloor \frac{n}{2} \rfloor} \ldots a_1)\}$. A node i can send REQUESTs to all nodes within time d_{max}. Because GRANTs from the farthest nodes will take d_{max} amount of time to return to node i, we obtain the simple result, $D = 2[\frac{n}{2}]$.

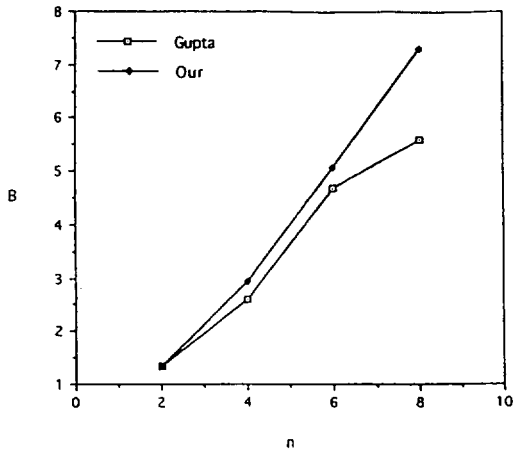
4.2 Average Blocking Delay

The blocking delay, B_{ij}, is defined for every pair of nodes (i, j) and measures the maximum number of sequential messages required after node j relinquishes the resource and before node i can access it; node i is supposed to be blocked while waiting for node j to relinquish the resource. The average blocking delay for a node i, B_i is defined as the average of B_{ij} over all $j \in Q_n$, $j \neq i$. Similarly, the average blocking delay for an algorithm, B, is defined as the average of B_i over all $i \in Q_n$.

It is easy to show that $B_{ij} = d$ for a node $j \in S_i^*$ where d is a distance between i and j since after j relinquishes the resource, it takes only d units of time for a GRANT to travel from j to i. On the other hand, $B_{ij} = 2 * [\frac{n}{2}]$ for node j S_i. Since our algorithm visits all $2 * \begin{pmatrix} [\frac{n}{2}] \\ d \end{pmatrix}$ nodes that are at distance of d from i, we obtain

$$B = \frac{1}{N-1} \left[\sum_{d=1}^{\lfloor \frac{n}{2} \rfloor} 2d \begin{pmatrix} [\frac{n}{2}] \\ d \end{pmatrix} + 2[\frac{n}{2}](N - 2^{\lfloor \frac{n}{2} \rfloor + 1} + 1) \right],$$

where N is the number of nodes in the n-dimensional hypercube. B_i will be the same for all i. Therefore $B = B_i$.



(Fig. 4) Comparison of blocking delay for Gupta et al.'s algorithm with our algorithm

4.3 Number of Messages per Access to Critical Resource

Let NM_i denote the number of messages generated during one resource access by node i. Obviously, NM_i will be minimum when there is no contention, but it will be maximized when there is contention at each node in S_i.

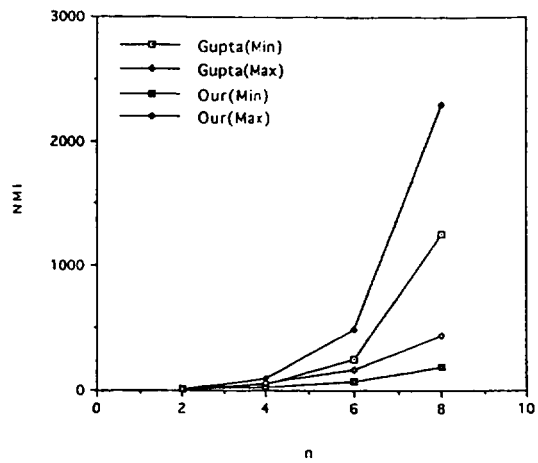
The minimum NM_i is achieved in the absence of any contention and corresponds to the shortest message exchange of (REQUEST, GRANT, RELEASE) of the three messages between i and all $k \in S_i^*$, $k \neq i$. Since in our algorithm, a node at distance d exchanges one REQUEST, one GRANT, and one RELEASE with the node i, a node at distance d will contribute three messages to Min NM_i.

The maximum NMi occurs in the presence of contention at each node k in S_i^*, $k \neq i$, and it corresponds to the most exchange of seven messages (REQUEST, INQUIRE, YIELD, new GRANT, FAIL, GRANT, RELEASE).

Since our algorithm started at node i traverses all $2 * \left\lfloor \begin{matrix} [\frac{n}{2}] \\ d \end{matrix} \right\rfloor$ nodes at distance d from i,

$$Min\ NM_i = \sum_{d=1}^{\lfloor \frac{n}{2} \rfloor} (3d * 2 * \left\lfloor \begin{matrix} [\frac{n}{2}] \\ d \end{matrix} \right\rfloor), \text{ and}$$

$$Max\ NM_i = \sum_{d=1}^{\lfloor \frac{n}{2} \rfloor} (7d * 2 * \left\lfloor \begin{matrix} [\frac{n}{2}] \\ d \end{matrix} \right\rfloor).$$



(Fig. 5) Comparison of Min NM_i and Max NM_i for Gupta et al.'s algorithm with our algorithm

5. Conclusion

We have presented an efficient decentralized, symmetric mutual exclusion algorithm for the hypercube architecture. Our algorithm has used the technique that can be embedded a mesh into a hypercube. For the performance of our algorithm, first the minimum round-trip delay is equal to that of Gupta et al.'s algorithm, secondly the average blocking delay is a little longer than that of Gupta et al.'s algorithm, thirdly the number of messages per access to critical resource is fewer than that of Gupta et al.'s algorithm. A node ($a_n a_{n-1} \ldots a_2 a_1$), once it know the dimension of the hypercube, can locally compute the sets of ($a_n \ldots a_{\lfloor \frac{n}{2} \rfloor + 1} * \ldots *$) and ($* \ldots * a_{\lfloor \frac{n}{2} \rfloor} \ldots a_1$) to carry out the broadcast. Similarly to

Gupta et al.'s algorithms, our algorithm is best suited for real-time applications as well as easy to implement.

References

[1] A. Gupta, S. C. Bruell, and S. Ghosh, "Mutual Exclusion on a Hypercube," Journal of Parallel and Distributed Computing 17, pp. 327-336, 1993.

[2] V. Kumar, A. Grama, A. Gupta, and G. Karypis, Introduction to Parallel Computing. The Benjamin/Cummings Pub. Co., Inc., 1994.

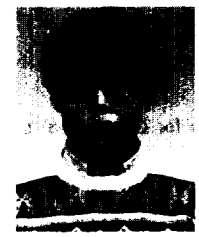
[3] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," Comm. of the ACM, Vol. 21, No. 7, pp. 558-565, July 1978.

[4] M. Maekawa, "A Sqrt(N) Algorithm for Mutual Exclusion in Decentralized Systems," ACM Trans. Computer Systems, Vol. 3, No. 2, pp. 145-159, May 1985.

[5] P. Ramanathan and K. G. Shin, "Reliable Broadcast in Hypercube Multicomputers," IEEE Trans. Computer Systems, Vol. 37, No. 12, pp. 1654-1657, Dec. 1988.

[6] G. Ricart and A. Agrawala, "An Optimal Algorithm for Mutual Exclusion in Computer Networks," Comm. ACM, Vol. 24, No. 1, pp. 9-27, Jan. 1981.

[7] B. A. Sanders, "The Information Structure of Distributed Mutual Exclusion Algorithms," ACM Trans. on Computer Systems, Vol. 5, No. 3, pp. 284-299, August 1987.



배 인 한

1984년 2월 경남대학교 전자계산학과 졸업(공학사)
1986년 2월 중앙대학교 대학원 전자계산학과 졸업(이학석사)
1990년 8월 중앙대학교 대학원 전자계산학과 졸업(공학박사)
1996년 2월~1997년 2월 오하이오 주립대 박사후 과정
1989년 3월~현재 대구효성가톨릭대학교 전자정보공학부 부교수
관심분야:분산시스템, 멀티미디어 시스템, 모빌 컴퓨팅 시스템