개선된 웜홀 라우터를 이용한 파이프라인 브로드캐스트

(Pipelined Broadcast with Enhanced Wormhole Routers)

전 민 수[†] 김 동 승[††]
(Minsoo Jeon) (Dongseung Kim)

요 약 이 논문은 n차원 하이퍼큐브에서 $O(m+n-1)$ 이내에 크기 m인 메시지를 브로드캐스트하는 파이프라인 브로드캐스트 (*pipelined broadcast*)를 제안한다. 이것은 도달가능집합(reachable set)으로부터 유도된 복제 나무(replication tree)를 이용한다. 이 브로드캐스트 방식은 $O(m\lceil n/\log(n+1)\rceil)$의 수행시간을 갖는 Ho-Kao의 알고리즘에 비해 성능이 크게 개선된 결과이다. 브로드캐스트 통신은 메시지 복제 기능을 갖는 all-port 웜홀 라우터를 이용한다. 이 논문은 알고리즘 기술 및 실제 구현시 이전 방식들과의 성능비교도 제시한다.

키워드 : 브로드캐스트, 웜홀 라우팅, 도달가능 집합, 중간 수신

Abstract This paper proposes the *pipelined broadcast* that broadcasts a message of size m in $O(m+n-1)$ time in an n-dimensional hypercube. It is based on the replication tree, which is derived from the reachable sets. It greatly improves the performance compared to Ho-Kao's algorithm with the time of $O(m\lceil n/\log(n+1)\rceil)$. The communication in the broadcast uses all-port wormhole router with message replication capability. This paper includes the algorithm together with performance comparisons to previous schemes in practical implementation.

Key words : broadcast, wormhole routing, reachable set, intermediate reception

1. Introduction

Message communication in parallel computers may be either point-to-point, with one source and one destination, or collective, with more than the two participants. In parallel processing it is often necessary to distribute common data fast and efficiently to multiple processors. In particular, message broadcast is one of such fundamental operations used in many scientific and engineering applications. Collective communications thus need to be properly designed according to their topology, router type, and routing policy of the target machines.

· 이 연구는 과학기술처 미래원천과제(98NF0304A01), 정보통신부 대학기초연구(97G-0620), 학술진흥재단 선도연구(1999-E00287)의 지원에 의해 수행되었음.

† 학생회원 : 고려대학교 전기공학과
msjeon@classic korea.ac.kr
†† 종신회원 : 고려대학교 전기공학과 교수
dkim@classic.korea ac kr

논문접수 : 2000년 6월 20일
심사완료 : 2001년 10월 8일

Most massively parallel computers available these days use the wormhole routing[1] in which a packet is divided into a number of *flits* (flow control bits). A flit is a basic unit of transmission, and it usually consists of a few bytes. The header flit governs the route, and the remaining flits follow it in a pipelined fashion[2,3]. Compared to the previous store-and-forward switching method[1], wormhole routing enables fast communication by minimizing the communication latency incurred when a message passes through intermediate processors (or *nodes*)[3].

Wormhole routers control flit reception and transmission in a fast way. Each router is connected to its local processor/memory by one or more pairs of internal links. One link of each pair is for input, the other for output. If each node possesses exactly one pair of internal links, then it has a *one-port* router. In one-port architecture the processor must transmit (receive) messages sequentially to (from) its neighbors. In an *all-port* system, however, every external link has a corresponding pair of internal links, thus the processor can send to (receive from) the neighbors through all external links simultaneously[4].

To improve collective communication performance two functions are added to the wormhole router: message replication (later on the abbreviation MR is used) and intermediate reception (IR)[4]. MR is a hardware feature that allows a router to replicate incoming flits and send the copy to an output port simultaneously. IR is a feature that allows router to deliver an incoming flit to the processor while it forwards the flit to its neighbor. IR is used to deliver a message efficiently to multiple inter mediate nodes on its way to a specified destination [5]. Based on the enhanced routers this paper develops a fast method to broadcast messages in a hypercube computer.

The remainder of the paper is organized as follows. Section 2 presents the devised broadcast for collective communication. Analysis and comparisons are given in Sect. 3, and a conclusion is given in Sect. 4.

2. Fast broadcast

A broadcast is a communication function to distribute a common message to all nodes in a parallel computer. Consider the broadcast in a hypercube computer. A few known algorithms that perform broadcast in hypercubes are *one-port* spanning binomial-tree (SBT) algorithm[6], and Ho and Kao's *all-port* broadcast algorithm (abbreviated as HK-broadcast)[7]. They require $O(mn)$ and $O(m\lceil n/\log(n+1)\rceil)$ time, respectively, to broadcast a message of size m in an n-dimensional hypercube which consists of $N=2^n$ nodes. Their times are *products* where one of the multipliers is a function of n. While only a single port does message transmission/reception at a time in a one-port router, messages arrive/depart simultaneously through all of its ports concurrently in all-port architecture. In HK-broadcast each node sends out received message to neighbor nodes at a time *after* it finishes the reception of the *whole message*. In the pipelined broadcast, however, each node does not wait for the complete message arrival. It instead transmits the message flit (partial message) to its neighbors as soon as it arrives at the node.

HK-broadcast is developed for all-port wormhole-routed hypercubes. The algorithm uses the concept of a dimension-simple path to recursively divide the network into subcubes of nearly equal size. A path $P = v_0, v_1, \cdots, v_d$ in an n-cube is called *dimension-simple* if there exists a sequence $i_1, i_2, \cdots, i_d$ of distinct cube dimensions such that for all $v_j, j \geq 1$, v_j is obtained from v_{j-1} by complementing the bit at dimension i_j. The path P is called *ascending* if $i_1 < i_2 < \cdots < i_d$. For example, an ascending dimension-simple path from node 0000000 in a 7-cube is

$$0000000 \rightarrow 0000001 \rightarrow 0000011 \rightarrow 0000111 \rightarrow 0001111 \rightarrow 0011111 \rightarrow 0111111 \rightarrow 1111111$$

In an all-port wormhole-routed hypercube in which dimension-ordered routing is performed by resolving addresses from top to bottom, one node can send a message to all nodes along such a path simultaneously (some staggering may occur due to message start-up latency). In this manner, an n-cube can be partitioned into $n+1$ subcubes such that each subcube contains one node on the dimension-simple path. Fig. 1 depicts the operation of the HK-broadcast algorithm in a 7-cube.

The pipelined broadcast in a hypercube is based on the *replication tree*, which is derived from the *reachable sets*[8] as explained below. Let S be a source (origin) node of a broadcast in an n-dimensional hypercube (or n-cube). The binary representation of S is $(s_{n-1}s_{n-2}\cdots s_0)$, where $s_k=\{0,1\}$ for all $k=0,1,\cdots,n-1$. A *subcube* of n-cube is a hypercube with smaller dimensions, which consists of all nodes with the same prefix in their binary index. The reachable set of S with respect to the dimension d_i, denoted by $R_i^n(S)$, $i \in \{0,1,\cdots,n\}$, is the subcube $(s_{n-1}s_{n-2}\cdots s_{i+1}\overline{s_i}\, x_{i-1}\cdots x_0)$ where x_k represents don't care. $R_n^n(S)$ is

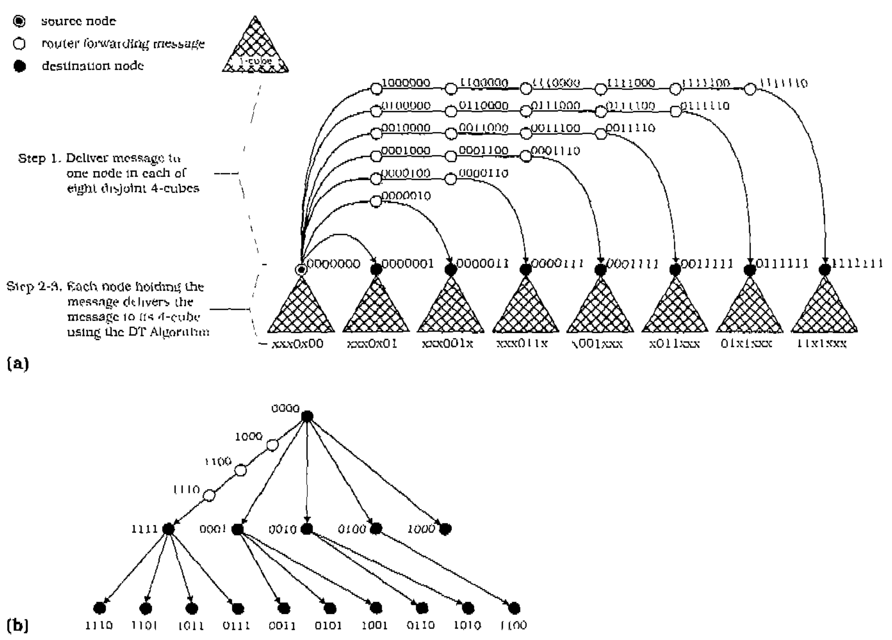


Fig. 1 Ho-Kao broadcast algorithm as executed on an all-port 7-cube: (a) overall algorithm and (b) two-step broadcast in a 4-cube.

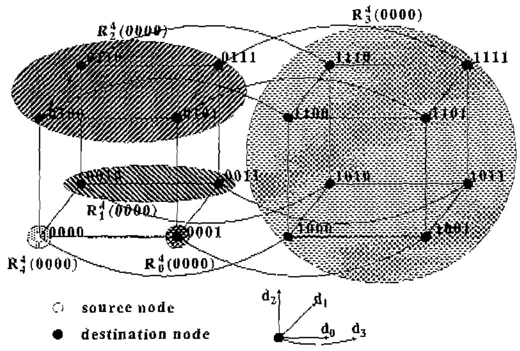


Fig. 2 Reachable sets of S(0000) in a 4-dimensional hypercube

S itself. $R^n(S)$ represents all reachable sets of node S, i.e. $R^n(S)$ = $\{R^n_0(S),\ R^n_1(S),\ \cdots, R^n_n(S)\}$. A *leader* node of $R^n_i(S)$ is $(s_{n-1}\cdots s_{i+1}\overline{s_i}s_{i-1}\cdots s_0)$, and it is an immediate neighbor of S along dimension d_i. Leader nodes of $R^n(S)$ are n neighbor nodes of S. Reachable sets $R^n_i(S)$, $i\in\{0,1,\cdots,n\}$, for any given source S, are disjoint one another, and the union of all reachable sets is the n-cube. For an illustration, Fig. 2 shows all the reachable sets $R^4_i(0000)$, for $i=\{0,1,2,3,4\}$ with S=0000 in a four-dimensional hypercube. $R^4_4(0000)$ is the source node 0000 itself. 1000, 0100, 0010, and 0001 are leader nodes of $R^4_3(0000)$, $R^4_2(0000)$, $R^4_1(0000)$, and $R^4_0(0000)$, respectively.

A *replication tree* for the pipelined broadcast is constructed from the reachable sets. The root of the tree at level 0 is the source(S) of the broadcast. Children of S at level 1 are all leader nodes of the reachable sets $R^n_i(S)$, $i=0,1,\cdots,n-1$. Since each reachable set is also a hypercube (or a subcube of an n-cube), new reachable sets (called sub-reachable sets) and the corresponding leaders can be obtained. For example, let S'_{n-1} be the new source of the subcube $R^n_{n-1}(S)$. Then, $R^{n-1}_0(S'_{n-1})$, $R^{n-1}_1(S'_{n-1})$, $\cdots,R^{n-1}_{n-1}(S'_{n-1})$ are new reachable sets, and n-1 leaders are those who have the same binary index as S'_{n-1} except one bit. (In the subcube of $R^4_3(0000)$,

$S'_3=1000$, and leaders are 1001, 1010, and 1100.) The nodes at level 2 of the tree are leaders of the current reachable sets. This process of computing nodes at deeper levels continues recursively until a complete replication tree is built (refer to Fig. 3). Fig. 4 shows a replication tree for a 4-cube with the source S=0000.

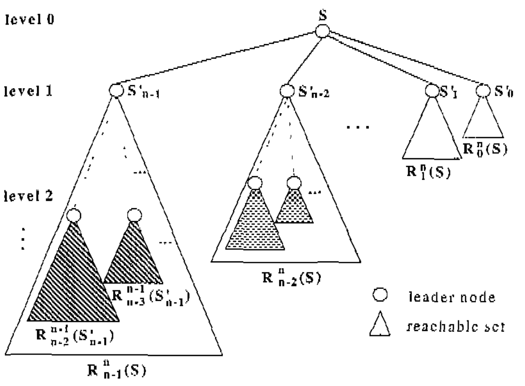


Fig. 3 Conceptual diagram of building a replication tree

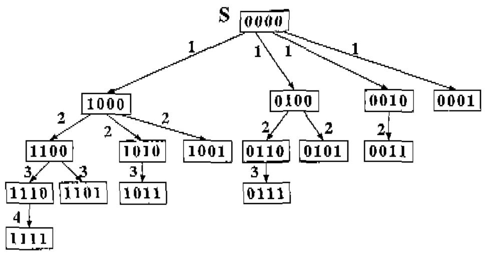


Fig. 4 Replication tree for a 4-cube with S=0000

Suppose there is an m-flit message, enclosed by a *header* and a *tail* flits. The tail is an end-of-message flit. In pipelined broadcast, the header consists of a pair of data (*type*, *address*). The type is either R(regular) or B(broadcast). If it is R, it is for a regular point-to-point wormhole routing, and the router sends incoming flit to one of its neighbors toward the destination *address*. Type B is for broadcast, for which the *address* is the index of the root of replication tree, and the all-port router in the current node replicates each incoming flit and gives out simultaneously to each of its children. The pipelined broadcast progresses in a sequence given by the replication tree. The order of message propagation relies on the levels as marked on the edges of the replication tree in Fig. 4. The broadcast starts at the root by sending a message flit to all its neighbor nodes at a time using the all-port router. Flits flow out one by one from the root. All non-leaf nodes that receive flits immediately copy and pass them to their descendents. They send out flits instead of waiting for the remainder of the message. All children of a parent receive the same flit at a time. The process continues until they pass the last flit, i.e. the tail flit. This is why the scheme is called *pipelined* broadcast.

It is important to find the execution time for the broadcast. Let's call the farthest node from the root the *terminal node* (For example, in Fig. 4, given S=0000, 1111 is the terminal node.). The broadcast ends when the terminal node receives the tail flit. Assume that there is no delay in replication of flits in the router. It needs n units of time for the header to arrive at the terminal node. After the header on, one flit arrives per unit time, thus, it takes another $m-1$ units until it receives the tail flit. By summing them up the broadcast requires $O(n+m-1)$ units of time. This reduces the time enormously from $O(m\lceil n/\log(n+1)\rceil)$ of HK-algorithm. For a long message the required time becomes $O(m)$, *independent* of the size of the hypercube, since $m \gg n$.

A formal description of the broadcast is given in Fig. 5 in semi-programming language. Algorithm A gives the procedure executed at the source node. This algorithm uses the function **send**(*dest*, *source*, *data*), which represents a message transmission from the *source* node (or the root node) to *dest* node. **forall** is a construct that executes the next statement simultaneously for all indices. Algorithm B is executed at each destination node, where **last_one**(S) returns the position index of the rightmost 1 in an n-bit address. Here bits are numbered from right to left. For example, **last_one**(1011000) returns 3. **replicate&forward**(*dest*,

Algorithm A: Broadcast at source node

```
Input: Message data and source index S.
Output: Send data to n outgoing ports with a format
    (destination, source, data). data are sent in flits.
Procedure:
    begin
        forall $i$ = 0 to $n$-1
            send( $S\oplus2^i$ , S, data)
    end
```

```
Algorithm B: Broadcast at destination nodes
Input: Incoming message (T, S, data), where
    T is the current node address ( $t_{n-1}\cdots t_1t_0$ )
    S is the source of the broadcast ( $s_{n-1}\cdots s_1s_0$ )
Output: When a message flit arrives. until the tail comes in,
    replicate-and-forward the flit data to output ports.
Procedure:
    begin
        if ( $(s_0\oplus t_0)\neq1$ )
            forall { $0\le i\le$ last_one( $S\oplus T$)-1}
                replicate&forward( $T\oplus2^i$, S, data) to Port$_i$
    end
```

Fig. 5 Pipelined broadcast algorithm

source, data) is a function that receives, replicates, and sends *data* to *dest*.

We may also think that a straightforward broadcasting is possible using IR by sending a message along a Hamiltonian path, from the source node to the end node of the path, in $O(m)$ time. The delay incurred in IR is assumed zero in this case. In practice, if we consider any amount of the delay per node (ever if it is a constant time), it would be $O(2^n+m-1)$ in an n-dimensional hypercube. In pipelined broadcast, links occupied during one broadcast can not be used by others until it completes. To avoid a possible deadlock when several nodes attempt to broadcast at the same time, the broadcast should be performed one by one using synchronization among nodes. The synchronization overhead of $O(\log n)$ is negligible in most cases.

3. Implementation consideration

The time complexity of the pipelined broadcast obtained above assumed zero-time flit replication and no message-communication overhead. Practically, they may not be ignored. Below the performance reflecting their effect is evaluated and compared to the previous results.

The latency of point-to-point message transmission of m-bytes with distance d was modeled as $T_s+dL_f/B+m/B$, where T_s is the startup time, L_f is the length of each flit, and B is the bandwidth of the link[9]. The total time for HK-algorithm in an n-dimensional hypercube is reported as

$$T_{HK}(m,n)=\alpha_nT_s+\beta_nT_f+\gamma_nT_c \qquad (1)$$

where $T_f=L_f/B$ is the cost to transmit a header flit, $T_c=1/B$ that for a nonheader flit, and $\alpha_n,\beta_n,\gamma_n$ are derived recursively as follows[10]:

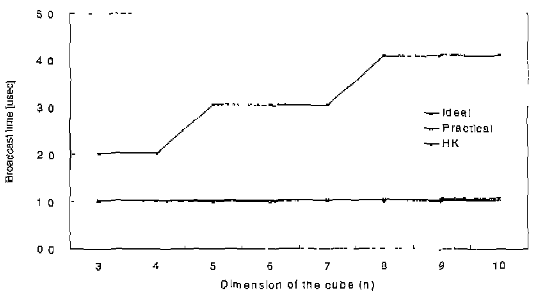
$$\begin{aligned}\alpha_n = \gamma_n &= 1+\alpha_{n-\lfloor\log_2 n+1\rfloor}, \quad n\ge5\\ \beta_n &= n+\beta_{n-\lfloor\log_2 n+1\rfloor}, \quad n\ge5\\ \alpha_1&=1,\ \alpha_i=\gamma_i=2, \quad 2\le i\le4\\ \beta_1&=1,\ \beta_2=2,\ \beta_3=4,\ \beta_4=5\end{aligned}$$

The execution of pipelined broadcast is estimated by reflecting the accumulated replication cost from the root to the terminal node where T_r is the cost of unit replication:

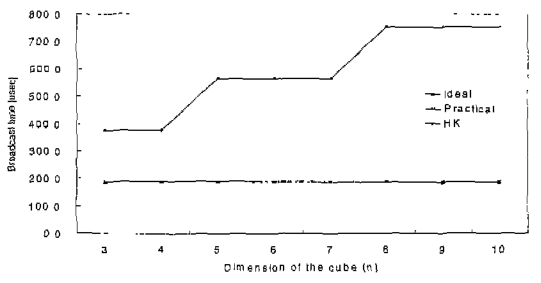
$$T_{pipe\_ideal}(m,n)=T_s+nT_f+mT_r+nT_r \qquad (2)$$

For the simple comparison purpose, let's assume that $L_f=1$ and $T_r=T_f$. Then $T_c=T_r=T_f$ and the Equation 2 becomes

$$T_{pipe\_practical}(m,n)=T_s+mT_f+2nT_f \qquad (3)$$



(a) 4-byte message



(b) 64-Kbyte message

Fig. 6 Broadcast times on Cray T3E with respect to the machine size($t_s=1\mu$sec, $t_f=0.0029\mu$sec)

Execution times computed by Equations 1, 2, and 3 are plotted in Fig. 6, for the messages of 4 bytes and 64K bytes, which represent small and large messages, respectively. We adopt the parameters of Cray T3E, which are $T_s = 1\mu\text{sec}$, $T_f = 0.0029\mu\text{sec}$ [11]. Notice that the scales of y-axis are different. As shown in Fig. 6, the time by the pipelined broadcast is quite smaller even with the replication cost, and almost independent of the size of the hypercube. Other parameter values such as Intel Paragon, IBM SP2, NCUBE-2, and Intel Delta produce similar results to those of Cray T3E.

4. Concluding Remarks

We have presented a fast broadcast scheme using all-port wormhole routers with the intermediate message reception and the message replication. The speed enhancement in our broadcast is due to pipelining of message transmission to the farthest node while intermediate nodes both accept and relay. In the pipelined broadcast, once the header reaches a destination, the remaining part of the message arrives successively in a pipelined manner. The message traveling the longest path spends the greatest time of $O(n+m-1)$ in an n-cube where m is the message size. The broadcast time is linear, no more a product of m and n. We believe many other collective communications can also be improved using the same idea.

References

[1] K. Hwang and Z. Xu, *Scalable parallel computing: technology, architecture, programming*, McGraw Hill, 1998.

[2] X. Lin and L.M. Ni, "Deadlock-free multicast wormhole routing in multicomputer networks," *Proc. 18th Int'l Symp. Computer Architecture*, pp. 116-125, 1991.

[3] L. M. Ni and P. K. McKinley, "A survey of wormhole routing techniques in direct networks," *IEEE Computer*, vol. 26, no. 2, pp. 62-76, 1993.

[4] P. K. Mckinley and Y. Tsai, "Collective communication in wormhole-routed massively parallel computers," *IEEE Computer*, pp. 39-50, 1995.

[5] D Kim and S-H Kim, "$O(\log n)$ numerical algorithms on a mesh with wormhole routing," *Information Processing Letters*, pp. 129-136, 1994.

[6] H. Sullivan and T. R. Bashkow, "A large scale, homogeneous, fully distributed parallel machine," *Proc. the 4th Annu. Symp. Computer Architecture*, vol. 5, pp. 105-124, 1977.

[7] C.-T. Ho and M. Kao, "Optimal broadcast in all-port wormhole-routed hypercubes," *IEEE Trans. Parallel and Distributed Systems*, Vol. 6, No. 2, Feb. 1995, pp. 200-204.

[8] V. Halwan and F. Özgüner, "Efficient multicast algorithms in all-port wormhole-routed hypercubes," *Proc. the 1997 Int'l Conf. on Parallel Processing*, pp. 84-91, 1997.

[9] Z. Xu and K. Hwang, "Modeling communication overhead: MPI and MPL performance on the IBM SP2," *IEEE Transactions on Parallel and Distributed Technology*, vol. 4, no. 1, 1996.

[10] Y. Tseng, S. Wang, and C. Ho, "Efficient broadcasting in wormhole-routed multicomputers: a network-partitioning approach," *IEEE Trans. on Parallel and Distributed Systems*, vol. 10, no. 1, pp. 44-61, Jan. 1999.

[11] Performance of the Cray T3E multiprocessor, http://www.sgi.com/t3e/performance .html, 1999.

김 동 승

1978년 서울대학교 전자공학 학사. 1980년 한국과학원 전기및전자공학 석사. 1988년 University of Southern California 컴퓨터공학 박사. 1980년 ~ 1983년 경북대학교 전임강사. 1988년 ~ 1989년 University of Southern California Post-doc 연구원. 1989년 ~ 1995년 포항공과대학 조,부교수. 1995년 ~ 현재 고려대학교 교수. 관심분야는 클러스터 컴퓨팅, 알고리즘, 분산/병렬 처리

전 민 수

1996년 고려대학교 전기공학 학사. 1998년 고려대학교 메카트로닉스 석사. 1998년 ~ 현재 고려대학교 전기공학 박사과정 재학중. 관심분야는 분산/병렬 처리, 알고리즘, 컴퓨터 구조