Journal of Marine Science and Technology

Volume 7
Issue 2 *12/15/2009*

Article 7

Spanning Trees for Binary Directed DE Bruijn Networks and Their Applications to Load Balancing

Ming-Bo Lin
Associate Professor, Department of Electronic Engineering, National Taiwan University of Science and Technology, 43, Keelung Road Section 4, Taipei, Taiwan

Ming-Hong Bai
Research assistant, Institute of Information Science, Academia Sinica

Gene Eu Jan
Associate Professor, Department of Computer Science, National Taiwan Ocean University, Keelung, Taiwan.

SPANNING TREES FOR BINARY DIRECTED DE BRUIJN NETWORKS AND THEIR APPLICATIONS TO LOAD BALANCING

Ming-Bo Lin,* Ming-Hong Bai,** and Gene Eu Jan***

Key words: de Bruijn network, downward spanning tree, load balancing, load sharing, parallel computer systems, and upward spanning tree.

ABSTRACT

One of the major reasons of using parallel computer systems is that they have the potential for improving performance and resource sharing. To achieve this, an efficient way must be provided to broadcast a message or messages from a node to every other nodes in the system. However, the efficiency of transferring messages in a system is determined by the architecture of the underlying interconnection network of the system. In this paper, we consider the systems based on binary directed de Bruijn networks and define two shortest path spanning trees: the upward-0 spanning tree and the downward-0 spanning tree, to meet various message transfer requirements. To demonstrate the usage of these spanning trees, an application to the load-balancing problem is considered. The resulting time complexity is $O(\log^2 N + \Sigma_{\forall \Delta i \neq 0} \Delta_i)$, where N is the number of nodes and Δ_i is the total transfer time for the load difference of each node i, for all $1 \leq i \leq N$, on the binary directed de Bruijn networks.

INTRODUCTION

Direct networks have been extensively used in highly parallel computer systems. One of these is the hypercube network, whose properties have been extensively studied in the literature [4]. The hypercube network belongs to the kinds of unbounded-degree networks. Recently, the researches have been motivated to the bounded-degree networks[8, 9].

One of the most popular bounded-degree networks is the de Bruijn network that is based on the de Bruijn graph [4, 6, 8, 11]. The attractive properties of a d-ary directed or undirected de Bruijn graphs with N nodes are that each node is of degree $2d$ and there are Nd edges. For simplicity and practical considerations, in this paper, we only consider the binary directed de Bruijn networks.

One major reason to use parallel computer systems is that they have the potential for improving performance and resource sharing[3]. For the latter, an efficient way must be provided to broadcast a message or messages from a node to every other nodes in the system. Two shortest path spanning trees: the upward-0 spanning tree and the downward-0 spanning tree, are defined in this paper to fulfill this requirement.

To demonstrate the usage of these spanning trees, an application to solving the load-balancing problem on the binary directed de Bruijn networks is discussed. The load-balancing problem usually raises from parallel computer systems since it is possible for some processors to be heavily loaded while others to be lightly loaded or even idle. To maximize the performance of such systems, it is necessary to keep every processor busy while some tasks are waiting for service in the systems. Two distinct strategies have been proposed [7, 10, 12, 13] for this purpose. Load-balancing algorithms explore the possibility of equalizing the workload among the processors while load-sharing algorithms simply attempt to assure that no processor is idle while some tasks are waiting for service.

In general, load-balancing algorithms require many more resources from the systems than do load-sharing algorithms [2]. Therefore, the extra resource requirement may outweigh the potential benefits of load balancing if we do not have a good enough load-balancing algorithm.

Load balancing can be viewed as a search for appropriate pairings among processors that are heavily loaded and those that are lightly loaded [1]. Three issues are intimately related to load balancing. They are: load difference evaluation, that is, to classify the processors as overloaded, balanced, and underloaded, mapping between overloaded and underloaded processors, and the redistribution of the load among the processors. Of course, the communication overhead, which depends on the communication mechanisms supported by the underlying parallel computer system,

Paper Received February 2, 1999. Revised October 13, 1999. Accepted December 16, 1999. Author for Correspondence: M. B. Lin.

**Associate Professor, Department of Electronic Engineering, National Taiwan University of Science and Technology, 43, Keelung Road Section 4, Taipei, Taiwan.*

***Research assistant, Institute of Information Science, Academia Sinica.*

**** Associate Professor, Department of Computer Science, National Taiwan Ocean University, Keelung, Taiwan.*

associated with load transfers must be minimized. In this paper, we assume that the basic workload unit is a task and that all tasks are independent, that is, they can be assigned independently to any processor and obtain the same result.

Based on the proposed shortest path spanning trees, the resulting time complexity for the load-balancing algorithm is $O(\log^2 N + \Sigma_{\forall \Delta i} = 0\ \Delta_i)$, where N is the number of nodes and Δ_i is the total transfer time for the load difference of each node i, for all $1 \le i \le N$, on the binary directed de Bruijn network.

The rest of the paper is organized as follows. Section 2 reviews and establishes some useful features of the binary directed de Bruijn networks that will be used throughout the paper. In this section, we also define two spanning trees. Section 3 describes how to apply the spanning trees to the load-balancing problem for the systems based on the binary directed de Bruijn networks. Section 4 proves the correctness and analyzes the performance of the load-balancing algorithm. The paper is then concluded in Section 5.

BINARY DIRECTED DE BRUIJN NETWORKS

In this section, we define the k-dimensional binary directed de Bruijn graph and network, routing schemes, downward and upward spanning trees, and derive some important properties of these spanning trees. These properties will be used in the load-balancing algorithm to be introduced in the next section.

Basic Definitions and Routing Schemes

In what follows, we define the k-dimensional binary directed de Bruijn network and its routing schemes.

Definition 1 *A binary k-dimensional directed de Bruijn graph, denoted as DDB(k), consisting of* 2^k *nodes and* 2^{k+1} *directed edges, is defined as:* $DDB(k) = (V_k, E_k)$, *where* $V_k = \{0, 1, 2 \ldots, 2^{k-1}\}$ *and* $E_k = \{<S, T> | T = 2S \bmod 2^k$, *for all* $S, T \in V_k\} \cup \{<S, T> | T = (2S \bmod 2^k) + 1$, *for all* $S, T \in V_k\}$.

Let $X = x_k x_{k-1} \ldots x_1$, where $x_i \in \{0, 1\}$, be a node on a *DDB*(k), then it is connected to two other nodes: $x_{k-1} x_{k-2} \ldots x_1 0$ and $x_{k-1} x_{k-2} \ldots x_1 1$, which are called left-child node and right-child node, respectively. The edges connected to node $x_{k-1} x_{k-2} \ldots x_1 0$ and $x_{k-1} x_{k-2} \ldots x_1 1$ are called 0 channel and 1 channel, respectively. An example of a *DDB*(3) graph along with the depictions of its 0 and 1 channels is shown in Fig. 1. The 0 channels are shown in plain lines while the 1 channels in bold lines.

The parallel computer system based on the *DDB* (k) is called a de Bruijn network and denoted by the

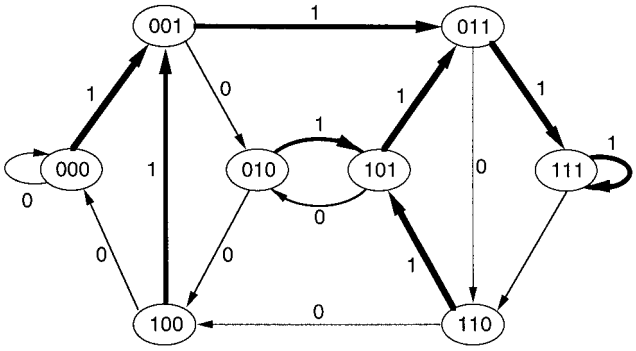


Fig. 1. An example of 3-dimensional binary directed de Bruijn graph.

DDB(k) network on which the nodes are composed of processors and the edges are the communication links between processors.

The definition of the *DDB*(k) network shows that the address relationship between a node and its two child nodes is a shift operation. Therefore, we define two shift operations: ShiftLeft and ShiftRight, as follows. Let node $X = x_k x_{k-1} \ldots x_1$ be an arbitrary node on a *DDB* (k) network and $b \in \{0, 1\}$. Then

$$ShiftLeft\ (x_k x_{k-1} \ldots x_1, b) = x_{k-1} x_{k-2} \ldots x_1 b$$

$$ShiftRight\ (x_k x_{k-1} \ldots x_1, b) = b x_k x_{k-1} \ldots x_2$$

It is easy to show that two parent nodes, P_0 and P_1, of a node X on a *DDB*(k), have node addresses: $P_0 = ShiftRight\ (X, 0)$ and $P_1 = ShiftRight\ (X, 1)$, respectively. In addition, two child nodes, Y_0 and Y_1, of node X, have the node addresses: $Y_0 = ShiftLeft\ (X,0)$ and $Y_1 = ShiftLeft\ (X, 1)$, respectively.

The following lemma finds a routing path between any two nodes on a *DDB*(k) network.

Lemma1 *Let* $X = (x_k x_{k-1} \ldots x_1)$ *and* $Y = (y_k y_{k-1} \ldots y_1)$ *be any two nodes on a DDB(k) network, then* $P = \{X_k, X_{k-1}, \ldots, X_1\}$ *is a path between these two nodes, where* $X_i = ShiftLeft\ (X_{i+1}, y_i)$, *for all* $1 \le i \le k$ *and* $X_{k+1} = X$. *For convenience, we usually represent P as* $x_k x_{k-1} \ldots x_1 y_k y_{k-1} \ldots y_1$, *where each node on the path is composed of a k-bit window from left to right.*

Since the number of shifts is equal to k, the path length of P is always $k = \log_2 N$, where N is the number of nodes of the *DDB*(k) network, and therefore we usually call this as the length-k path. The routing method based on this is called the length-k routing scheme. The following lemma will establish the foundation for another routing method called the optimal routing scheme [5].

Lemma 2 *Let* $X = (x_k x_{k-1} \ldots x_1)$ *and* $Y = (y_k y_{k-1} \ldots y_1)$

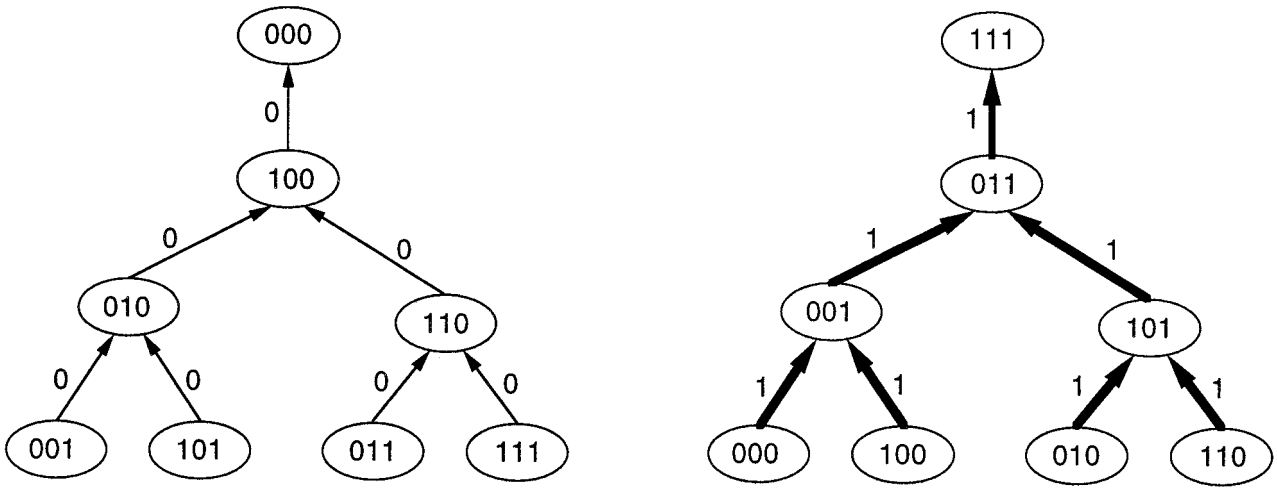


Fig. 2. The examples of G_{UT0} and G_{UT1} for a *DDB*(3) network.

be any two nodes on a DDB(k) network and $P = \{X_{k-c}, X_{k-c-1}, \ldots, X_1\}$ *be a path on the network, where* $X_i = ShiftLeft\ (X_{i+1}, y_i)$, *for all* $1 \le i \le k - c$, *and* $X_{k-c+1} = X$. *c is defined as*
$c = max\ \{s | 0 \le s \le k, x_s x_{s-1} \ldots x_1 = y_k y_{k-1} \ldots y_{k-s+1}\}$
then P is the shortest path between X and Y with length $k - c$. *For notational simplicity, we usually represent P as* $x_k x_{k-1} \ldots x_1 y_{k-c} y_{k-c-1} \ldots y_1$.

Upward-0 and Upward-1 Spanning trees

The following defines the upward-0 and upward-1 spanning trees of a *DDB*(*k*) network, respectively.

Definition 2 *Let* $G_{UT0} = (V_k, E_{UT0})$ *be a subnetwork of a DDB(k) network, where* $V_k = \{0, 1, 2, \ldots, 2^k - 1\}$ *and* $E_{UT0} = \{<u, v> | <u, v>$ *are all of the* 0 *channels except for the edge* <00 … 0, 00 … 0> *of DDB*(*k*)}. *Similarly, let* $G_{UT1} = (V_k, E_{UT1})$ b*e a subnetwork of a DDB(k) network, where* $V_k = \{0, 1, 2, \ldots, 2^k - 1\}$ *and* $E_{UT1} = \{<u, v> | <u, v>$ *are all of the 1 channels except for the edge* <11 … 1, 11 … 1> *of DDB*(*k*)}.
Excluding edges <00 … 0, 00 … 0} and <11 … 1, 11 … 1> in the above definition is necessary to avoid forming cycles in the subnetworks $G_{UT0} = (V_k, E_{UT0})$ and $G_{UT1} = (V_k, E_{UT1})$, respectively. Thus, this makes it possible for them to be as spanning trees as stated in the following theorem. Examples of G_{UT0} and G_{UT1} for a *DDB*(3) network are shown in Fig. 2 (a) and (b), respectively.

Theorem 1 $G_{UT0} = (V_k, E_{UT0})$ *and* $G_{UT1} = (V_k, E_{UT1})$ *are spanning trees of the DDB*(*k*) *network with roots at nodes* 00 … 0 *and* 11 … 1, *respectively*.

Proof: To prove the subnetwork $G_{UT0} = (V_k, E_{UT0})$ is a spanning tree of a *DDB*(*k*) network, we first prove it is connected. Lemma 1 implies that each node can reach node 00 … 0 through a series of ShiftLeft operations with 0 filling. More precisely, any node can arrive at node 00 … 0 through a series of 0 channels. Consequently, the connected property is valid. Next, the subnetwork $G_{UT0} = (V_k, E_{UT0})$ is acyclic since it only contains $N - 1$ links, where N is the number of nodes of the *DDB*(*k*) network, because edge 00 … 0, 00… 0 is excluded. As a consequence, $G_{UT0} = (V_k, E_{UT0})$ is a spanning tree of the *DDB*(*k*) network. Similarly, it is easy to prove the subnetwork $G_{UT1} = (V_k, E_{UT1})$ is also a spanning tree of the *DDB*(*k*) network. □
The tree G_{UT0} is called the upward-0 spanning tree since it consists of 0 channels only and these channels are directed upward to root node 00 … 0. Similarly, G_{UT1} is called the upward-1 spanning tree with root node 11 … 1. It is easy to show that both the upward-0 and upward-1 spanning trees of a *DDB*(*k*) network are unique.

The following theorem establishes some useful properties of both G_{UT0} and G_{UT1}.

Theorem 2 *The upward-*0 *and upward-*1 *spanning trees,* G_{UT0} *and* G_{UT1}, *of a DDB(k) network have the following properties:*

1. *Every node can reach node* 00 … 0 *on* G_{UT0} *and node* 11 … 1 *on* G_{UT1}, *respectively, with the maximal path length* $\log_2 N$, *where N is the number of nodes of the DDB(k) network.*

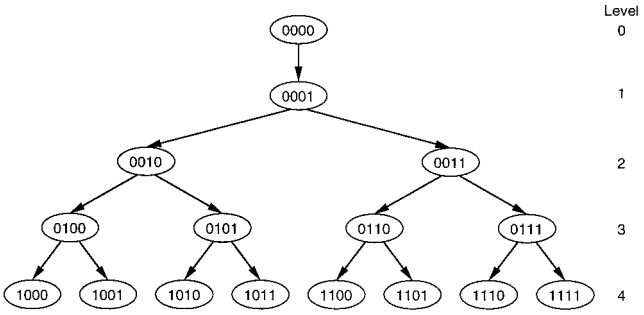


Fig. 3. The downward-0 spanning tree of the *DDB*(4) network.

2. *All external nodes (i.e., leaves) are odd-numbered on* G_{UT0} *and even-numbered on* G_{UT1}, *respectively.*

3. *All internal nodes are even-numbered on* G_{UT0} *and odd-numbered on* G_{UT1}, *respectively.*

4. *Each internal node X except for the root node on both* G_{UT0} *and* G_{UT1} *is connected from two other nodes with addresses ShiftRight* (*X*, 0) *and ShiftRight* (*X*, 1), *respectively.*

Note that the upward-0 and upward-1 spanning trees are isomorphic in structure so that in the rest of this paper, we only consider the upward-0 spanning tree.

Downward-0 and Downward-1 Spanning trees

Another kind of spanning trees are downward-0 and downward-1 spanning trees. These spanning trees are defined as follows.

Definition 3 *Let* $G_{DT0} = (V_k, E_{DT0})$ *be a subnetwork of a DDB(k) network, where* $V_k = \{0, 1, 2, \ldots, 2^k - 1\}$ *and* $E_{DT0} = \{<u, v> | v = ShiftLeft\ (u, 0)$ *and* $v > u$, *for all* $u, v \in V_k\} \cup \{<u, v> | v = ShiftLeft\ (u,1)$ *and* $v > u$, *for all* $u, v\ V_k\}$. *Similarly, let* $G_{DT1} = (V_k, E_{DT1})$ *be a subnetwork of a DDB*(*k*) *network, where* $V_k = \{0, 1, 2, \ldots, 2^k - 1\}$ and $E_{DT1} = \{<u, v> | v = ShiftLeft\ (u, 0)$ *and* $v > u$, *for all* $u, v \in V_k\} \cup \{<u, v> | v = ShiftLeft\ (u,1)$ *and* $v > u$, *for all* $u, v\ V_k\}$.

The following theorem states that G_{DT0} and G_{DT1} are spanning trees of the *DDB*(*k*) network.

Theorem 3 $G_{DT0} = (V_k, E_{DT0})$ *and* $G_{DT1} = (V_k, E_{DT1})$ *are spanning trees of the DDB*(*k*) *network with roots at nodes* 00 … 0 *and* 11 … 1, *respectively.*

Usually, G_{DT0} is called the downward-0 spanning tree since it starts from root node 00 … 0 and downward to leaf nodes. Similarly, G_{DT1} is called the downward-1 spanning tree with root node 11 … 1. An example showing the downward-0 spanning tree of a *DDB*(4) network is depicted in Fig. 3.

The following theorem establishes some useful properties of both G_{DT0} and G_{DT1}.

Theorem 4 The downward-0 and downward-1 spanning trees, G_{DT0} and G_{DT1}, of a *DDB*(*k*) network have the following properties.

1. *Every node can be reached from root nodes* 00 … 0 *on* G_{DT0} *and* 11 … 1 *on* G_{DT1}, *respectively, with the maximal path length* $\log^2 N$, *where N is the number of nodes of the DDB(k) network.*

2. *The node address of any node other than the root with label l is* $\geq 2^{l-1}$ *on* G_{DT0} *and* $\geq N - 2^l$ *on* G_{DT1}, *where l is labeled starting with* 0 *from the root node and N is the number of nodes of the DDB(k) network.*

3. *All external nodes on* G_{DT0} *have node addresses* $\geq \frac{N}{2}$; *all external nodes on* G_{DT1} *have node addresses* $< \frac{N}{2}$.

4. *The parent of any node X of* G_{DT0} *has the node address ShiftRight* (*X*, 0); *the parent of any node X of* G_{DT1} *has the node address ShiftRight* (*X*, 1).

Like the upward-0 and upward-1 spanning tree, the downward-0 and downward-1 spanning trees are isomorphic in structure . Hence, in the rest of this paper, we only consider the downward-0 spanning tree because it has the same root node as the upward-0 spanning tree.

The following corollary follows immediately.

Corollary 1 *Let* X_1 *and* X_2 *be any two nodes on the downward-0 spanning tree of a DDB*(*k*) *network with the same level l, where* $2 \leq l \leq k$, *then* X_1 *and* X_2 *have the same address part* $x_k x_{k-1} \ldots x_l$.

LOAD-BALANCING ALGORITHM

In this section, we apply the spanning trees defined in the previous section to solve the load-balancing problem for the *DDB*(*k*) networks.

In general, a load-balancing algorithm consists of four major parts: load difference evaluation, load collection, task reassignment, and load redistribution. For convenience, we assume that each task is an independent unit and may be executed by any processor on the system.

The load-balancing algorithm for a *DDB*(*k*) network is shown in the following.

Algorithm: LoadBalancing

This algorithm is used to balance the loads of each node on the *DDB*(*k*) network.

Input: Unbalanced loads of each node on the *DDB*(*k*) network.

Output: The load difference of each node on the *DDB* (*k*) network is within ± 1 unit.
begin
1: Load difference evaluation: Evaluate load difference among nodes on the *DDB*(*k*) network.

1.1: Sum up the load from each node on the network using the upward-0 spanning tree, that is, compute $L = \sum_{i=1}^{N} l_i$, where l_i, for all $1 \le i \le N$, is the load of node i.

1.2: Broadcast the average of load $AVG = \left\lfloor \frac{L}{N} \right\rfloor$ to every other node from root node 00 … 0 by using the downward-0 spanning tree.

1.3: Each node determines itself is a balanced ($\Delta_i = 0$), overloaded ($\Delta_i > 0$), or underloaded ($\Delta_i < 0$) node by computing $\Delta_i = l_i - AVG$, where $1 \le i \le N$.
2: Load collection: The root node collects the extra tasks from each other node of the network by using the upward-0 spanning tree.
3: Task reassignment: Reassign tasks of each node on the network using the downward-0 spanning tree.

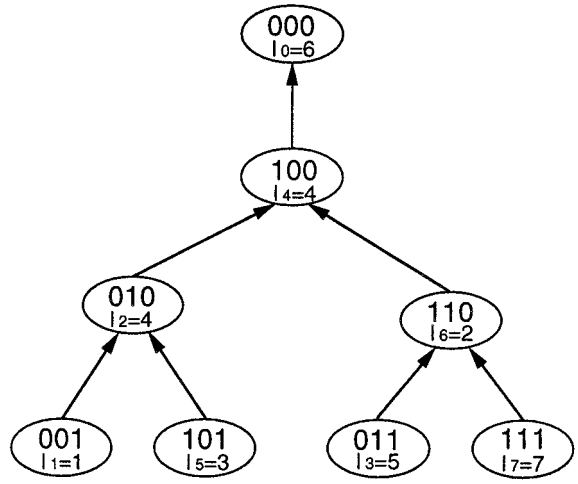
3.1: Each node receives the load requests $load_0$ and $load_1$ from its left and right children, respectively.

3.2: Each node sends its extra load ($load_0 + load_1 + myload - AVG$) to its parent node.
4: Load redistribution: The root node distributes the extra tasks to each other node of the network by using the downward-0 spanning tree.
end {End of LoadBalancing algorithm.}

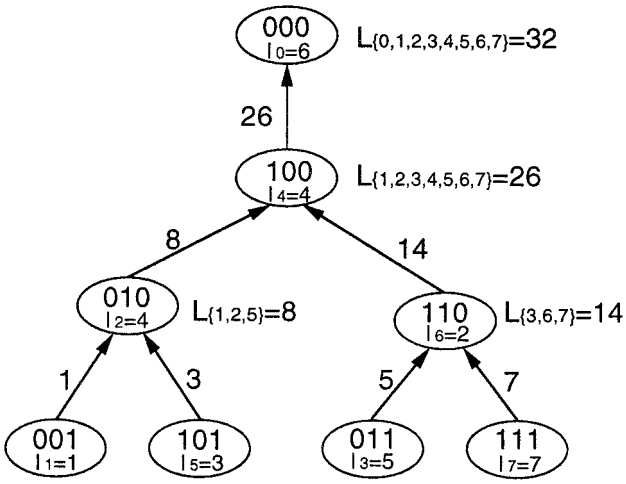
Due to the similarity of different uses of both the upward-0 spanning tree and the downward-0 spanning tree in the above load balancing algorithm, in what follows we will describe only two examples of them. One is SumofLoad (*myid*) which uses the upward-0 spanning tree for summing up the load from each node and is described as follows.

Procedure: SumofLoad (myid)

Input: Loads of each node on a *DDB*(*k*) network.
Output: The summation of the loads from each node is collected at root node 00 … 0.
begin
if IsOdd (*myid*) then send *myload* to ShiftLeft (*myid*, 0);
else begin
sum = *myload*;
receive $load_0$ from ShiftRight (*myid*, 0);
sum = *sum* + $load_0$;
receive $load_1$ from ShiftRight (*myid*, 1);
sum = *sum* + $load_1$;
if (*myid* ≠ 0) then send *sum* to ShiftLeft (*myid*, 0);
end {end of else}
end {End of SumofLoad procedure.}



(a) The load distribution before the execution of SumofLoad procedure.



(b) The load summation after the execution of SumofLoad procedure.

Fig. 4. An example to illustrate the operations of SumofLoad procedure using the upward-0 spanning tree.

As shown in Fig. 4(a) is the state before the execution of SumofLoad procedure while Fig. 4(b) is the results of every iteration of executing procedure SumofLoad.

The other is AVGBroadcast that is used to broadcast the *AVG* computed by the root node to every other node on the *DDB*(*k*) network using the downward-0 spanning tree.
Procedure: AVGBroadcast (*myid*)
Input: The AVG of load computed by root node 00 … 0 on a *DDB*(*k*) network.
Output: Every node has the AVG.
begin

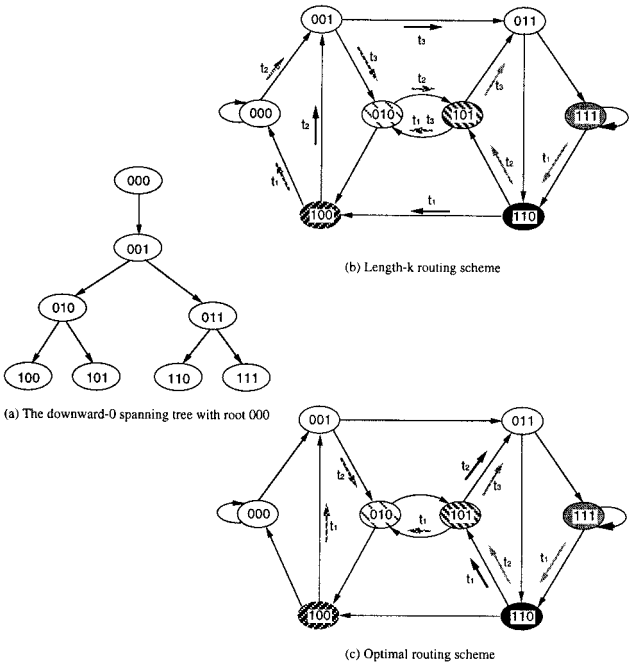


Fig. 5. Illustrations of edge disjoint and temporal edge disjoint paths on the downward-0 spanning tree of a *DDB*(3) network.

if *myid* = 0 then send *AVG* to ShiftLeft (*myid*, 1);
else if *myid* < $\frac{N}{2}$ then
receive *AVG* from ShiftRight (*myid*, 0);
send *AVG* to ShiftLeft (*myid*, 0);
send *AVG* to ShiftLeft (*myid*, 1);
else receive *AVG* from ShiftRight (*myid*, 0);
end {End of AVGBroadcast procedure.}

The reason why we could not use the same upward-0 spanning tree as that for SumofLoad procedure to broadcast *AVG* to every other node on the *DDB*(*k*) network is that the underlying *DDB*(*k*) network is directed so that every link or channel can carry message in only one direction. Thus, in order to broadcast a message as fast as possible the downward-0 spanning tree is used.

PERFORMANCE AND CORRECTNESS

In this section we prove the correctness and analyze the performance of Load balancing algorithm described in the previous section.

The correctness of LoadBalancing algorithm is established by the following theorem.

Theorem 5 *After the LoadBalancing algorithm terminates, the load difference* Δ_i *of each node* X_i *is within* $\pm$ 1 *unit of tasks, where* $1 \le i \le N$ *and N is the number of nodes of the DDB*(*k*) *network.*

Proof: The load difference evaluation step guarantees the average of load *AVG* can be computed and broadcast to every node on the *DDB*(*k*) network since the upward-0 and downward-0 spanning trees are used. The load collection step collects all extra tasks of each node on the upward-0 spanning tree at root node 00 … 0. Since the spanning tree is used, no node can be excluded to carry out the operation. The extra tasks collected at the root node then redistribute to every underloaded node using the downward-0 spanning tree. To assure this, the task reassignment step must be performed before the load redistribution step. At the task reassignment step, we use the same downward-0 spanning tree as for the load redistribution step. Hence, through the execution of task reassignment, each node on the downward-0 spanning tree knows how many loads are needed by its subtree and this information is also broadcast up to its parent node. The root node 00 … 0 can redistribute the extra tasks to its child nodes and each node can then distribute the extra tasks received from its parent node to its child nodes according to the information that it has recorded. Therefore, each node guarantees to receive the required load from its parent node and enters into the balanced state, that is, $\Delta_i = \pm 1$, where $1 \le i \le N$. □

To estimate the time complexity of LoadBalancing algorithm, we need to analyze the complexity of task reassignment. As described before, the major operation of task reassignment is to propagate the load request from leaf nodes to its parent node one by one along the downward-0 spanning tree up to the root node 00 … 0. The bottleneck of this operation is that there is no direct connection from a node to its parent node. Consequently, the length-*k* routing scheme or the optimal routing scheme is needed for routing the information from nodes to their parent nodes.

As a first glance, it seems that many nodes will contest edges when they transfer load request messages up to their parent nodes. However, as the illustration shown in Fig. 5 (b) with the length-*k* routing scheme, nodes 111 and 110 will transfer messages to their parent node 011 using paths 111 → 110 → 101 → 011 and 110 → 100 → 001 → 011, respectively. There is no common edge of these two paths. Thus, they are edge-disjointed. Another case is shown in Fig. 5 (c). In this case, assume that the optimal routing scheme is used. Nodes 111 and 110 will transfer messages to their parent node 011 using paths 111 → 110 → 101 → 011 and 110 → 101 → 011, respectively. The subpath 110 → 101 → 011 is common to these two paths. Thus, they are not edge-disjointed paths. However, in fact nodes 111 and 110 use this subpath at different time as the timestamps shown in the figure. Consequently, these paths can be considered as edge-disjointed paths if the timestamp is added to them for scheduling their usage.

Since the path length in the worst case of the optimal routing scheme is the same as that of the length-k routing scheme, in the rest of this paper we will not further consider this routing scheme. The following lemma establishes the result that any two paths from two nodes at the same level $l + 1$ to their parent nodes on the downward-0 spanning tree of a given $DDB(k)$ network are edge-disjointed if the length-k routing scheme is used.

Lemma 3 *Assuming that the length-k routing scheme is used, any two paths from X_1 to Y_1 and from X_2 to Y_2 on the downward-0 spanning tree of a given DDB(k) network are edge-disjointed, where X_1 and X_2 are two arbitrary nodes at the same level $l + 1$ and Y_1 and Y_2 are the parent nodes of X_1 and X_2, respectively.*

Proof: Let P_{ath1} and P_{ath2} be any two paths starting from nodes X_1 and X_2 at level $l + 1$ to their parent nodes Y_1 and Y_2 at level l on the downward-0 spanning tree. As the implication of Corollary 1, X_1 and X_2 have the same address part from $(l + 1)$th to kth bit while Y_1 *and* Y_2 have the same address part from lth to kth bit. In addition, *ShiftRight* $(X, 0)$ is the node address of parent node of X. Hence, the P_{ath1} and P_{ath2} may be represented as follows.

$$Path_1 \leftrightarrow x_k x_{k-1} \ldots x_{l+1} 0 x_{l-1} \ldots x_i \ldots x_1 0 x_k x_{k-1} \ldots x_{l+1} 0 x_{l-1} \ldots x_{i+2} \ldots x_2$$

$$Path_2 \leftrightarrow x_k x_{k-1} \ldots x_{l+1} 1 x'_{l-1} \cdots x'_i \cdots x'_1 0 x_k x_{k-1} \cdots x_{l+1} x'_{l-1} \cdots x'_{i+2} \cdots x'_2$$

Without loss of generality, assume that the first possible identical node is started with ith bit, i.e., $N = x_i \ldots x_1 0 x_k x_{k-1} \ldots x_{l+1} 0 x_{l-1} \ldots x_{i+2}$ for $Path_1$ and $N' = x'_i \cdots x'_1 0 x_k x_{k-1} \cdots x_{l+1} 1 x'_{l-1} \cdots x'_{i+2} \cdots x'_2$ for $Path_2$, where $1 \le i \le l - 1$. Nodes N and N' could not be the same because their lth bits are different, one is 0 and the other is *1*. The only possibility that they are identical is as $i = l - 1$. However, we could not find the next identical node in this case since the next bit, which is the lth bit, of node N is 0 and node N' is 1. These 0 and 1 bits will comprise the node addresses of both $Path_1$ and $Path_2$ thereafter. Hence, both paths can only have one identical node. That is, there is no edge conflict of $Path_1$ and $Path_2$.

□

An example is shown in Fig. 3, as nodes 1011 and 1111 are routed to their parent nodes 0101 and 0111, respectively, using the length-k routing scheme, the paths from 1011 to 0101 and from 1111 to 0111 are:

$$1011 \rightarrow 0110 \rightarrow 1101 \rightarrow 1010 \rightarrow 0101$$
$$1111 \rightarrow 1110 \rightarrow 1101 \rightarrow 1011 \rightarrow 0111,$$

respectively. Consequently, they are edge-disjointed although node 1101 is used by both paths.

Having this result, we may establish the time complexity of procedure TaskReassignment as follows.

Lemma 4 *The time complexity of Task reassignment is $O(\log^2 N)$, where N is the number of nodes on the DDB (k) network.*

Proof: Due to that the depth of downward-0 spanning tree is $\log_2 N$ and the maximum length of the routing path from level $l + 1$ to l is at most $\log_2 N$ if the length-k routing scheme is used, the running time of Task reassignment is $O(\log^2 N)$. □

Theorem 6 *The running time of LoadBalancing algorithm is $O(\log^2 N + \Sigma_{\forall \Delta i \neq 0} \Delta_i)$, where N is the number of nodes on the DDB(k) network and Δ_i is the number of load difference, for all $1 \le i \le N$. Here we assume that the transfer time of each task is one time unit.*

Proof: It is easy to compute the expected running time of LoadBalancing algorithm. The load difference evaluation step uses SumofLoad and AVGBroadcast procedures to sum up the load difference of each node and broadcast the the average of load *AVG* to every node, respectively. Both of these two procedures need $O(\log N)$ time since both the upward-0 spanning tree used by SumofLoad procedure and the downward-0 spanning tree used by AVGBroadcast procedure have $O(\log N)$ depth. The expected running time of load collection step is $O(\log N + \Sigma_{\forall \Delta i > 0} \Delta_i)$, where $O(\log N)$ is contributed by the upward-0 spanning tree and the $\Sigma_{\forall \Delta i > 0} \Delta_i$ is the upper bound of message transfer time for collecting all extra tasks in the system at root node 00 … 0. As for the task reassignment step, Lemma 4 gives the time bound $O(\log^2 N)$. The load redistribution step has the similar time bound as that for load collection step and is $O(\log N + \Sigma_{\forall \Delta i < 0} \Delta_i)$. As a consequence, the total running time is $O(\log^2 N + \Sigma_{\forall \Delta i \neq 0} \Delta_i)$.

CONCLUSION

In this paper two shortest path spanning trees, one is called upward-0 spanning tree and the other is called downward-0 spanning tree, for binary k-dimensional directed de Bruijn networks are defined and applied to solve the load-balancing problem for the systems based on the $DDB(k)$ networks. The resulting load-balancing algorithm has the time complexity of $O(\log^2 N + \Sigma_{\forall \Delta i \neq 0} \Delta_i)$, where N is the number of nodes and Δ_i is the total transfer time of load difference of each node i, for all $1 \le i \le N$, on the $DDB(k)$ network, respectively.

REFERENCES

1. Gene Eu Jan and Ming-Bo Lin, "Effective load balancing on highly parallel multicomputers based on superconcentrators," *Proceedings of the International Conference on Parallel and Distributed Systems 1994*, pp. 216-221, Dec. (1994).
2. Orly Kremien and Jeff Kramer, "Methodical analysis of adaptive load sharing algorithms," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 3, No. 6, pp. 747-760, Nov. (1992).
3. Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis," *Introduction to Parallel Computing: Design and Analysis of Algorithms*," Redwood City, California: The Benjamin/Cummings Publishing Company, Inc., (1994).
4. F. Thomson Leighton,"*Introduction to Parallel Algorithms and Architectures: Arrays, Trees, and Hypercubes*, " San Mateo, California: Morgan Kaufmann Publishers, (1992).
5. Zhen Liu and Ting-Yi Sung, "Routing and Transmitting Problems in de Bruijn Networks," *IEEE Trans. Comput.*, pp. 1056-1062, September (1996).
6. Z. Liu, "Optimal routing in the de Bruijn networks," *Proceedings of the* 10*th International Conference on Distributed Computing Systems*, pp. 537-544, June (1990).
7. R. Mirchandaney, D. Towsley, and J. A. Stankovic, "Analysis of the effetcs of delays on load sharing," *IEEE Trans. Comput.*, vol. C-38, no. 11 pp. 1513-1525, Nov. 1989.
8. M. R. Samatham and D. K. Pradham, "The de Bruijn multiprocessor network: a versatile parallel processing and sorting network for VLSI," *IEEE Transactions on Computers*, Vol. 38, No. 4, pp. 567-581, April (1989).
9. Eric J. Schwabe, "On the computational equivalence of hypercube-derived networks," *Proceedings of the* 2*nd Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 388-397, July (1990).
10. K. G. Shin and Y. C. Chang, "Load sharing in distributed real time systems with state-change broadcasts," *IEEE Trans. Comput.*, vol. C-38, no. 8 pp. 1124-1142, Aug. (1989).
11. Kumar N. Sivarajan and Rajiv Ramaswami, "Lightwave networks based on de Bruijn graphs," *IEEE Transactions on Networking*, Vol. 2, No. 1, pp. 70-79, February (1994).
12. Y. T. Wang and R. J. T. Morris, "Load sharing in distributed systems," *IEEE Trans. Comput.*, vol. C-34, no. 3 pp. 204-217, Mar. (1985).
13. Marc H. Willebeek-LeMair and Anthony P. Reeves, "Strategies for dynamic load balancing on highly parallel computers," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 4, No. 9, pp. 979-993, Sept. (1993).

二元有向de Bruijn網路之跨越樹及其在負載平衡上的應用

林銘波

國立台灣科技大學電子工程系副教授

白明弘

中央研究院資訊研究所研究助理

詹景裕

國立海洋大學資訊科學系副教授

摘　要

使用平行計算機系統的主要理由之一爲其具有改進計算性能與資源共用的潛力，欲達到此目的系統中必須有一個有效的方法將一個或是多個訊息由一個節點傳遞到其他節點上，然而此傳遞訊息的方法是否有效往往決定於所用的網路結構。在本論文中，我們將考慮二元有向de Bruijn網路並且定義兩個跨越樹分別稱爲：向上-0跨越樹(upward-0 spanning tree)與向下-0跨越樹(downward-0 spanning tree)，以滿足所需的訊息傳遞，並且應用於負載平衡(load balancing)。結果顯示，時間雜度爲$O(\log_2^2 N + \sum_{\forall \Delta_i \neq 0} \Delta_i)$，其中N爲節點數目而$\Delta_i$爲每一個節點$i$的資訊轉移時間，$1 \leq i \leq N$。