JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 11, 257-261 (1991)

Cost-Optimal Parallel Algorithms for Constructing 2-3 Trees*

BIING-FENG WANG AND GEN-HUEY CHEN

Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, Republic of China

In this paper, two cost-optimal parallel algorithms are presented for constructing 2-3 trees from sorted lists of data items. The two parallel algorithms are designed on a shared-memory SIMD computer; one, based on the EREW model, uses $N/\log\log N$ processors and requires $O(\log\log N)$ time and the other, based on the CREW model, uses N processors and requires $O(1)$ time, where N is the number of data items in the input sorted list.

1. INTRODUCTION

Search trees [6] such as binary search trees, m-way search trees, and 2-3 trees are efficient data structures for representing sorted lists. Search, insertion, and deletion with respect to a balanced search tree of N data items can be done in expected time $O(\log N)$. The problem of constructing a balanced search tree from a sorted list of N data items is to create a corresponding search tree with minimal height to represent the sorted list. To create such a tree, the data items kept in each node must be determined and the links between nodes must be set up properly.

Optimal sequential algorithms for constructing balanced binary search trees in $O(N)$ time can be found in [3, 4]. Recently, based on the EREW shared-memory model, Moitra and Iyengar [7, 8] have proposed parallel algorithms that use N processors to construct balanced binary search trees in $O(1)$ time. Further, based on the same model, Dekel *et al.* [5] have designed parallel algorithms that use N processors to construct balanced m-way search trees. The time complexities of Dekel *et al.*'s algorithms are $O(1)$ under an assumption that computing the kth, $0 \leq k \leq \log N$, power of m takes $O(1)$ time.

The main disadvantage of the m-way search tree is that it may become unbalanced when it is not static [1, 6], and thus logarithmic accessing time can not be guaranteed. For example, let us consider a sequence of updating operations, such as insertion and deletion, on a balanced m-way search tree. The tree may become unbalanced after these operations have been completed and then the accessing time of the tree may become proportional to the tree size in the worst case. Therefore, rebalancing is indispensable to an m-way search tree after some updating operations are performed on it. This leads to the unsuitability of on-line processing of the m-way search tree. As an alternative to the m-way search tree, some other trees have been proposed, for example, AVL tree [6], weight-balanced tree [6], and 2-3 tree [1]. Among them, the 2-3 tree is the most attractive one for the following reasons. First, rebalancing is unnecessary. Second, logarithmic accessing time is guaranteed for on-line processing. Third, accessing the 2-3 tree is easy to do. Fourth, as compared with the AVL tree and the weight-balanced tree, the 2-3 tree is free of additional information for keeping itself balanced. Finally, in addition to being an alternative representation of a sorted list, the 2-3 tree has the capabilities of concatenating and splitting lists. More description about the 2-3 tree can be found in [1, 2].

In this paper, we present two parallel algorithms for constructing 2-3 trees to represent sorted lists of data items. The two parallel algorithms are designed on the shared-memory SIMD (single instruction, multiple data) computer; one, based on the EREW (exclusive read, exclusive write) model, uses $N/\log\log N$ processors and requires $O(\log\log N)$ time, and the other, based on the CREW (concurrent read, exclusive write) model, uses N processors and requires $O(1)$ time, where N is the number of data items in the input sorted list. Both parallel algorithms are cost-optimal.

The rest of this paper is organized as follows. In the next section, the shapes of the constructed 2-3 trees are uniquely specified. Furthermore, some notations and definitions that are used throughout this paper are introduced. In Section 3, some properties of the constructed 2-3 trees are described. In Section 4, two cost-optimal parallel algorithms for constructing 2-3 trees are presented. Finally, concluding remarks are given in Section 5.

2. NOTATION AND DEFINITIONS

A *2-3 tree* is a tree in which every nonleaf node has two or three children and every path from the root to a leaf node is of the same length. A nonleaf node of the 2-3 tree owns one data item if it has two children and owns two data items if it has three children. A leaf node of the 2-3 tree owns one or two data items. Therefore, the number of data items that is contained in a 2-3 tree of height n ranges from $2^n - 1$ to

* This research is supported by the National Science Council of the Republic of China under Grant NSC-78-0408-E-002-02.

0743-7315/91 $3.00

$3^n - 1$ (we define the height of a tree as the number of its levels). The 2-3 tree can be used as a heap or as a search tree [1]. As a search tree, the data items contained in the 2-3 tree are arranged in such a way that if we traverse the 2-3 tree in in order traversal, we can get their sorted sequence. In this paper, we consider the 2-3 tree to be a search tree.

For the convenience of description, we use notation $\langle i, j \rangle$ to denote the jth node from the left at the ith level (see Fig. 1). A node $\langle i, j \rangle$ is said to be the *lowest common ancestor* of node $\langle i_1, j_1 \rangle$ and node $\langle i_2, j_2 \rangle$ if $\langle i_1, j_1 \rangle$ and $\langle i_2, j_2 \rangle$ are in different subtrees of $\langle i, j \rangle$. Moreover, if node $\langle i, j \rangle$ contains one data item k_1, then k_1 is said to be the *split data item* of node $\langle i_1, j_1 \rangle$ and node $\langle i_2, j_2 \rangle$. On the other hand, if node $\langle i, j \rangle$ contains two data items k_1 and k_2 (assume $k_1 < k_2$) and node $\langle i_1, j_1 \rangle$ and node $\langle i_2, j_2 \rangle$ are contained respectively in the left subtree and the middle subtree of $\langle i, j \rangle$, then k_1 is the split data item of $\langle i_1, j_1 \rangle$ and $\langle i_2, j_2 \rangle$. Likewise, k_2 is the split data item of $\langle i_1, j_1 \rangle$ and $\langle i_2, j_2 \rangle$ if they are contained in the middle subtree and the right subtree of $\langle i, j \rangle$, respectively.

Note that more than one 2-3 tree can represent the same sorted list. For example, Fig. 1 shows two 2-3 trees that represent the sorted list (1, 2, 3, . . . , 26, 27). Thus, it is necessary to specify the exact one that the proposed algorithms will construct. For a given sorted list of N data items, the uniquely constructed 2-3 tree has a minimal height $n = \lceil \log_3(N + 1) \rceil$. And there exists a unique pair of intergers c and l, where $1 \leq c \leq n$ and $l \geq 1$, such that all the nodes above the cth level own two data items, all the nonleaf nodes at and below the cth level own one data item, the leftmost l leaf nodes own two data items, and the other leaf nodes own one data item. For example, the 2-3 tree that the proposed algorithms will construct for the sorted list (1, 2, 3, . . . , 26, 27) is shown in Fig. 1a, where $c = 2$ and $l = 4$. The existence and uniqueness of the constructed 2-3 trees are proved in the next section.

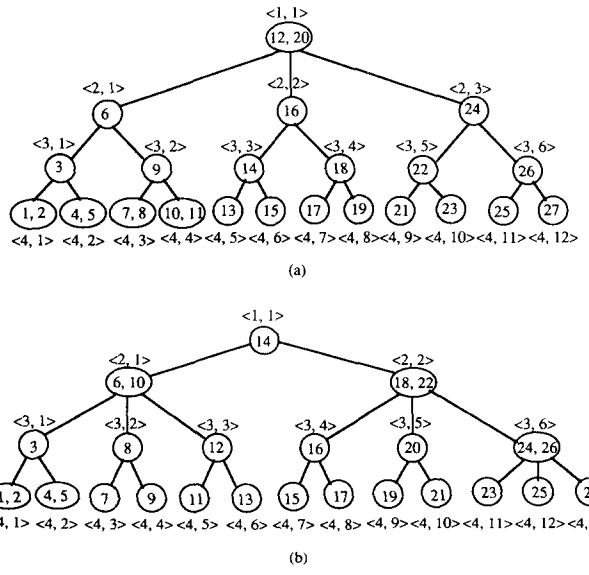

FIG. 1. Two different 2-3 trees that represent the sorted list (1, 2, 3, . . . , 26, 27).

In the following, we define notations that are used throughout this paper.

N: The size of the input sorted list. We assume $N > 3$.

T_N: The 2-3 tree that is constructed by the proposed algorithms for an input sorted list of size N.

n: The height of T_N, which is defined as the number of levels of T_N. Note that $n = \lceil \log_3(N + 1) \rceil$.

c: The marked level of T_N. That is, in T_N, all the nodes above the cth level own two data items and all the nonleaf nodes at and below the cth level own one data item.

l: The number of leaf nodes in T_N that own two data items. Note that $l \geq 1$.

V_N: The number of nodes in T_N.

$LCHILD_N(\langle i, j \rangle)(MCHILD_N(\langle i, j \rangle)$, $RCHILD_N(\langle i, j \rangle))$: The left (middle, right) child of node $\langle i, j \rangle$ in T_N. (If node $\langle i, j \rangle$ has only two children, then $MCHILD_N(\langle i, j \rangle)$ is undefined.)

$RANK1_N(\langle i, j \rangle)(RANK2_N(\langle i, j \rangle))$: The rank of the smaller (greater) data item of node $\langle i, j \rangle$ in the input sorted list. It equals the number of data items (inclusive of the smaller (greater) data item itself) that precede the smaller (greater) data item of node $\langle i, j \rangle$ in inorder traversal. (If node $\langle i, j \rangle$ owns only one data item, then $RANK2_N(\langle i, j \rangle)$ is undefined.)

$L1_N(\langle i, j \rangle)(L2_N(\langle i, j \rangle))$: The number of leaf nodes in T_N that precede the smaller (greater) data item of node $\langle i, j \rangle$ in in-order traversal. (If node $\langle i, j \rangle$ owns only one data item, then $L2_N(\langle i, j \rangle)$ is undefined.)

For example, let us consider Fig. 1a again, where $N = 27$, $n = 4$, $c = 2$, $l = 4$, $V_{27} = 22$, $LCHILD_{27}(\langle 2, 2 \rangle) = \langle 3, 3 \rangle$, $RCHILD_{27}(\langle 2, 2 \rangle) = \langle 3, 4 \rangle$, $MCHILD_{27}(\langle 2, 2 \rangle)$ is undefined, $RANK1_{27}(\langle 1, 1 \rangle) = 12$, $RANK2_{27}(\langle 1, 1 \rangle) = 20$, $L1_{27}(\langle 1, 1 \rangle) = 4$, and $L2_{27}(\langle 1, 1 \rangle) = 8$. The lowest common ancestor of node $\langle 4, 4 \rangle$ and node $\langle 4, 5 \rangle$ is node $\langle 1, 1 \rangle$ and their split data item is 12.

3. SOME PROPERTIES OF THE CONSTRUCTED 2-3 TREES

In this section we first prove that T_N uniquely exists and then introduce some properties of T_N which constitute the kernel of the proposed algorithms.

THEOREM 1. *T_N uniquely exists for $N > 3$.*

Proof. Note that according to the definition of T_N, T_N is uniquely specified by the pair c and l while the height of T_N is $n = \lceil \log_3(N + 1) \rceil$. Therefore, to complete the proof, we show that for an arbitrary N, there exists a unique pair of c and l satisfying the following equation and constraints:

$$N = 2*(1 + 3 + \cdots + 3^{c-2}) + 1*3^{c-1}*(1 + 2 + \cdots + 2^{n-c}) + l \quad (1)$$

$$1 \leq l \leq 3^{c-1}*2^{n-c} \quad (2)$$

$$1 \leq c \leq n. \quad (3)$$

Equation (1) means that the total number of data items contained in T_N equals N; constraint (2) means that l may not be greater than the number of leaf nodes.

Combining (1) and (2), we have

$$3^{c-1}*2^{n-c+1} - 1 < N \leq 3^c*2^{n-c} - 1, \quad (4)$$

from which

$$c - 1 < \log_{3/2}((N+1)/2^n) \leq c \quad (5)$$

can be derived.

Clearly, $\lceil \log_{3/2}((N+1)/2^n) \rceil$ is the unique solution of c that satisfies (5). To show that it also satisfies (3) is equivalent to showing the inequation

$$0 < \log_{3/2}((N+1)/2^n) \leq n. \quad (6)$$

Since $n = \lceil \log_3(N+1) \rceil$, we have

$$3^{n-1} \leq N \leq 3^n - 1,$$

which assures (6) for $N > 3$.

On the other hand,

$$l = N - 3^{c-1}*2^{n-c+1} + 1 \quad (7)$$

can be derived from (1). By combining (4) and (7), (2) can be proved. ■

COROLLARY 1. *For the uniquely constructed* 2-3 *tree* T_N, $n = \lceil \log_3(N+1) \rceil$, $c = \lceil \log_{3/2}((N+1)/2^n) \rceil$, $l = N - 3^{c-1}*2^{n-c+1} + 1$, *and* $V_N = (3^{c-1}*(2^{n-c+2} - 1) - 1)/2$.

According to the definition of T_N, the following lemma can be derived.

LEMMA 1. *For each node* $\langle i, j \rangle$ *in* T_N,

(a) *when* $1 \leq i < c$, $LCHILD_N(\langle i, j \rangle) = \langle i+1, 3*j-2 \rangle$, $MCHILD_N(\langle i, j \rangle) = \langle i+1, 3*j-1 \rangle$, *and* $RCHILD_N(\langle i, j \rangle) = \langle i+1, 3*j \rangle$,

(b) *when* $c \leq i < n$, $LCHILD_N(\langle i, j \rangle) = \langle i+1, 2*j-1 \rangle$, $MCHILD_N(\langle i, j \rangle)$ *is undefined, and* $RCHILD_N(\langle i, j \rangle) = \langle i+1, 2*j \rangle$, *and*

(c) *when* $i = n$, $LCHILD_N(\langle i, j \rangle)$, $MCHILD_N(\langle i, j \rangle)$, *and* $RCHILD_N(\langle i, j \rangle)$ *are all undefined.*

Let us consider a node $\langle i, j \rangle$ in T_N. If $1 \leq i < c$, the leaf nodes that precede the smaller (greater) data item of node $\langle i, j \rangle$ in inorder traversal are those that are contained in the subtrees with roots at $\langle i+1, 1 \rangle, \langle i+1, 2 \rangle, \ldots, \langle i+1, 3*j-2 \rangle$ ($\langle i+1, 1 \rangle, \langle i+1, 2 \rangle, \ldots, \langle i+1, 3*j-1 \rangle$), respectively. If $c \leq i \leq n$, the leaf nodes that precede the data item(s) of node $\langle i, j \rangle$ in inorder traversal can be determined similarly. Therefore, we have the following lemma.

LEMMA 2. $L1_N(\langle i, j \rangle) = (3*j-2)*3^{c-1-i}*2^{n-c}$ *if* $1 \leq i < c$, $(2*j-1)*2^{n-i-1}$ *if* $c \leq i < n$, *and* $j-1$ *if* $i = n$. *And,* $L2_N(\langle i, j \rangle) = (3*j-1)*3^{c-1-i}*2^{n-c}$ *if* $1 \leq i < c$, *undefined if* $c \leq i < n$ *or* ($i = n$ *and* $j > l$), *and* $j-1$ *if* $i = n$ *and* $j \leq l$.

Note that the shape of a constructed 2-3 tree is uniquely determined by the values of n and c. Since the leaf nodes of a constructed 2-3 tree may contain one or two data items, T_N may have the same shape for some different values of N. For example, let $N_{min} = N - l + 1$ and $N_{max} = N + 3^{c-1}*2^{n-c} - l$. $T_{N_{max}}$ ($T_{N_{min}}$) is the constructed 2-3 tree containing most (least) data items that has the same shape as T_N (that is, each leaf node in $T_{N_{min}}$, except for $\langle n, 1 \rangle$, contains one data item, and each leaf node in $T_{N_{max}}$ contains two data items). In the following, we introduce some properties of $T_{N_{max}}$.

LEMMA 3. *In* $T_{N_{max}}$, *the subtree with root at node* $\langle i, j \rangle$ *contains* $3^{c-i+1}*2^{n-c} - 1$ *data items if* $1 \leq i < c$ *and* $3*2^{n-i} - 1$ *data items if* $c \leq i \leq n$.

Proof. When $1 \leq i < c$, the number of data items that are contained in the subtree with root at node $\langle i, j \rangle$ is computed as the sum of $2*(1 + 3 + 3^2 + \cdots + 3^{c-i-1}) + 3^{c-i}*(1 + 2 + 2^2 + \cdots + 2^{n-c-1}) + 2*(3^{c-i}*2^{n-c})$, which can be simplified to $3^{c-i+1}*2^{n-c} - 1$. When $c \leq i \leq n$, the computation is similar. ■

Since the data items that precede the smaller data item of node $\langle i, 1 \rangle$ are those contained in the subtree with root at node $\langle i+1, 1 \rangle$, the following lemma can be derived from Lemma 3.

LEMMA 4. $RANK1_{N_{max}}(\langle i, 1 \rangle) = 3^{c-i}*2^{n-c}$ *if* $1 \leq i < c$, $3*2^{n-i-1}$ *if* $c \leq i < n$, *and* 1 *if* $i = n$.

LEMMA 5. $RANK1_{N_{max}}(\langle i, j+1 \rangle) = RANK1_{N_{max}}(\langle i, j \rangle) + 3^{c-i+1}*2^{n-c}$ *if* $1 \leq i < c$, $RANK1_{N_{max}}(\langle i, j \rangle) + 3*2^{n-i}$ *if* $c \leq i < n$, *and* $RANK1_{N_{max}}(\langle i, j \rangle) + 3$ *if* $i = n$.

Proof. The data items that are between the smaller data item of node $\langle i, j \rangle$ and the smaller data item of node $\langle i, j+1 \rangle$, while traversing $T_{N_{max}}$ in inorder traversal, include five parts: (i) the data items in the subtree with root at the middle child of $\langle i, j \rangle$, (ii) the greater data item of $\langle i, j \rangle$, (iii) the data items in the subtree with root at the right child of $\langle i, j \rangle$, (iv) the split data item of $\langle i, j \rangle$ and $\langle i, j+1 \rangle$, and (v) the data items in the subtree with root at the left child of $\langle i, j+1 \rangle$. Part (i) and part (ii) are empty if $i = n$

or $\langle i, j\rangle$ contains one data item. Part (iii) and part (v) are empty if $i = n$. By counting the above data items with the aid of Lemma 1 and Lemma 3, the lemma follows. ■

Similarly, we have the following lemma.

LEMMA 6. *$RANK2_{N_{max}}(\langle i, j\rangle) = RANK1_{N_{max}}(\langle i, j\rangle) + 3^{c-i}*2^{n-c}$ if $1 \leq i < c$, undefined if $c \leq i < n$, and $RANK1_{N_{max}}(\langle i, j\rangle) + 1$ if $i = n$.*

By Lemma 4, Lemma 5, and Lemma 6, $RANK1_{N_{max}}(\langle i, j\rangle)$ and $RANK2_{N_{max}}(\langle i, j\rangle)$ can be computed as follows.

THEOREM 2. *$RANK1_{N_{max}}(\langle i,j\rangle) = (3*j-2)*3^{c-i}*2^{n-c}$ if $1 \leq i < c$, $(2*j-1)*3*2^{n-i-1}$ if $c \leq i < n$, and $3*j-2$ if $i = n$. And, $RANK2_{N_{max}}(\langle i,j\rangle) = (3*j-1)*3^{c-i}*2^{n-c}$ if $1 \leq i < c$, undefined if $c \leq i < n$, and $3*j-1$ if $i = n$.*

Note that T_N and $T_{N_{max}}$ are of the same shape, but different in the number of data items that are kept in the leaf nodes. Only the leftmost l leaf nodes of T_N own two data items, while every leaf node of $T_{N_{max}}$ owns two data items. Therefore, we have the following theorem.

THEOREM 3. *$RANK1_N(\langle i, j\rangle) = RANK1_{N_{max}}(\langle i, j\rangle) - max\{0, L1_N(\langle i, j\rangle) - l\}$. And, $RANK2_N(\langle i, j\rangle) = RANK2_{N_{max}}(\langle i, j\rangle) - max\{0, L2_N(\langle i, j\rangle) - l\}$ if $1 \leq i < c$, undefined if $c \leq i < n$ or $(i = n$ and $j > l)$, and $RANK2_{N_{max}}(\langle i, j\rangle)$ if $i = n$ and $j \leq l$.*

4. PARALLEL CONSTRUCTION ALGORITHMS

In this section, we assume that T_N is to be stored in a linear array of length V_N. Before presenting the parallel construction algorithms, we need to specify a linear ordering for the nodes of T_N such that the node with order k, $1 \leq k \leq V_N$, is represented by the kth element of the linear array. A simple approach to do so is to number the nodes according to their breadth-first search order. For example, Table I shows the specified linear ordering for the nodes of the 2-3 tree that was depicted in Fig. 1a. Let $ORDER_N(\langle i, j\rangle)$ denote the order of node $\langle i, j\rangle$ in T_N and $NODE_N(k)$ denote the node with order k. According to the specification of the linear ordering, they can be determined as follows.

LEMMA 7. *$ORDER_N(\langle i, j\rangle) = (3^{i-1} - 1)/2 + j$ if $1 \leq i \leq c$ and $(3^{c-1}*(2^{i-c+1} - 1) - 1)/2 + j$ if $c < i \leq n$.*

LEMMA 8. *For $1 \leq k \leq V_N$, $NODE_N(k) = \langle i_k, j_k\rangle$, where $(i_k, j_k) = (\lceil \log_3(2*k+1)\rceil, k - (3^{i_k-1} - 1)/2)$ if $1 \leq k \leq (3^c - 1)/2$, and $(\lceil(\log_2((k - 3^{c-1})/(2*3^{c-1}) + 1)) + c\rceil, k - (3^{c-1}*(2^{i_k-c+1} - 1) - 1)/2)$ if $(3^c - 1)/2 + 1 \leq k \leq V_N$.*

Since each node of T_N is uniquely represented by an element of a linear array of length V_N, we have to determine for each element of the linear array which node of T_N it represents, which data items in the input sorted list it will keep, and which elements of the linear array are its children, in order to construct T_N. Clearly, these can be done independently for all the elements of the linear array. Thus, T_N can be constructed in parallel as follows: first, n, c, l, and V_N are computed according to Corollary 1; then, $RANK1_N(NODE_N(k))$, $RANK2_N(NODE_N(k))$, $ORDER_N(LCHILD_N(NODE_N(k)))$, $ORDER_N(MCHILD_N(NODE_N(k)))$, and $ORDER_N(RCHILD_N(NODE_N(k)))$ are computed for all $k = 1, 2, \ldots, V_N$ simultaneously according to Lemma 1, Lemma 7, Lemma 8, and Theorem 3.

Suppose $NODE_N(k) = \langle i_k, j_k\rangle$, where $1 \leq k \leq V_N$, and let $S_k = \{2^n, 3^{c-1}, 2^{n-c+1}, 2^{n-c+2}, 3^{i_k-1}, 2^{i_k-c+1}, 3^c, 3^{c-1-i_k}, 2^{n-c}, 2^{n-i_k-1}, 3^{c-i_k}\}$. Clearly, the time required to process the kth element is dependent on how fast the set S_k can be computed. In the following, we propose the parallel construction algorithms on the shared-memory SIMD computer under both the EREW model and the CREW model.

4.1. *Parallel Construction Algorithms Under the EREW Model*

A simple parallel algorithm for constructing T_N under the EREW model is to let each processor P_k, $1 \leq k \leq V_N$, process the kth element simultaneously. The time complexity is $O(\log \log N)$ $(=O(\log n))$, which is the time required for processor P_k to compute the set S_k. Since $V_N \leq N$ processors are needed, the parallel algorithm is not cost-optimal. However, a cost-optimal parallel algorithm that uses $V_N/\log \log N$ processors and runs in $O(\log \log N)$ time can be derived according to the following fact.

Fact 1. *Let $NODE_N(k) = \langle i_k, j_k\rangle$ and $NODE_N(k+1) = \langle i_{k+1}, j_{k+1}\rangle$. Then, $i_{k+1} - i_k = 0$ or 1. That is, $NODE_N(k)$ and $NODE_N(k+1)$ are two nodes at the same level or at the adjacent level.*

As a result of Fact 1, set S_{k+1} can be computed in additional constant time after set S_k has been computed. The cost-op-

TABLE I

The Specified Linear Ordering for the Nodes of the 2-3 Tree Depicted in Fig. 1a

Node $\langle i, j\rangle$	$\langle 1, 1\rangle$	$\langle 2, 1\rangle$	$\langle 2, 2\rangle$	$\langle 2, 3\rangle$	$\langle 3, 1\rangle$	$\langle 3, 2\rangle$	$\langle 3, 3\rangle$	$\langle 3, 4\rangle$	$\langle 3, 5\rangle$	$\langle 3, 6\rangle$	$\langle 4, 1\rangle$
Order k	1	2	3	4	5	6	7	8	9	10	11
Node $\langle i, j\rangle$	$\langle 4, 2\rangle$	$\langle 4, 3\rangle$	$\langle 4, 4\rangle$	$\langle 4, 5\rangle$	$\langle 4, 6\rangle$	$\langle 4, 7\rangle$	$\langle 4, 8\rangle$	$\langle 4, 9\rangle$	$\langle 4, 10\rangle$	$\langle 4, 11\rangle$	$\langle 4, 12\rangle$
Order k	12	13	14	15	16	17	18	19	20	21	22

timal parallel algorithm is simply to let each processor process log log N consecutive elements. Each processor processes the assigned elements in increasing order of their indices and therefore it takes $O(\log \log N)$ time for the first one and $O(1)$ time for each of the others.

THEOREM 4. *T_N can be constructed in $O(\log \log N)$ time on an EREW shared-memory SIMD computer with $V_N/\log \log N \leq N/\log \log N$ processors, which is cost-optimal.*

4.2. *Parallel Construction Algorithm Under the CREW Model*

Under the CREW model, T_N can be constructed in $O(1)$ time by the aid of two tables, *POWER_2*$[1..n]$ and *POWER_3*$[1..n]$, where *POWER_2*$[i]$ and *POWER_3*$[i]$, $1 \leq i \leq n$, store the values of 2^i and 3^i, respectively. The table *POWER_3*$[1..n]$ can be filled in by N processors as follows. Each processor P_k, $1 \leq k < N$, first computes $u_k = \lceil \log_3(k+1) \rceil$ and $l_k = \lfloor \log_3(k+1) \rfloor$ and fills *POWER_3*$[u_k]$ with $k+1$ if $u_k = l_k$. Then (since $3^{n-1} \leq N \leq 3^n - 1$), processor P_N fills *POWER_3*$[n]$ with $3 * POWER\_3[n-1]$. Table *POWER_2*$[1..n]$ can be filled in similarly. After the two tables have been established, T_N can be constructed in $O(1)$ time by letting each processor P_k, $1 \leq k \leq V_N$, process the kth element simultaneously. Now each set S_k can be computed in $O(1)$ time by looking up the tables.

THEOREM 5. *T_N can be constructed in $O(1)$ time on a CREW shared-memory SIMD computer with N processors, which is cost-optimal.*

5. CONCLUDING REMARKS

The advantages of the 2-3 tree over the binary and m-way search trees mainly come from the flexibility of its structure. But the flexibility also makes it difficult to design parallel algorithms for constructing it. In this paper, we have proposed two cost-optimal parallel algorithms for constructing 2-3 trees on the shared-memory SIMD computer under both the EREW model and the CREW model. Furthermore, it is not difficult to see that the proposed parallel algorithms can be adapted to a fixed number, say p, of processors, in which case the proposed parallel algorithms run in $O(N/p)$ time.

As an extension of this paper, the authors are now working on deriving parallel algorithms for constructing B-trees [9].

Received May 9, 1990; accepted October 8, 1990

REFERENCES

1. Aho, A. V., Hopcroft, J. E., and Ullman, J. D. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, MA, 1974.
2. Brown, M. R., and Tarjan, R. E. Design and analysis of data structures for representing sorted lists. *SIAM J. Comput.* **9,** 3 (Aug. 1980), 594–614.
3. Chang, H., and Iyengar, S. S. Efficient algorithm to globally balanced binary search trees. *Comm. ACM* **27,** 7 (July 1984), 695–702.
4. Day A. C. Balancing a binary tree. *Comput. J.* **19** (Nov. 1976), 360–361.
5. Dekel, E., Peng, S., and Iyengar, S. S. Optimal parallel algorithms for constructing and maintaining a balanced m-way search tree. *Int. J. Parallel Programming* **15,** 6 (1986), 503–528.
6. Knuth, D. E. *The Art of Computer Programming. Vol. 3. Sorting and Searching.* Addison-Wesley, Reading, MA, 1973.
7. Moitra, A., and Iyengar, S. S. A maximally parallel balancing algorithm for obtaining complete balanced binary trees. *IEEE Trans. Comput.* **C-34,** 6 (June 1985), 563–565.
8. Moitra, A., and Iyengar, S. S. Derivation of a parallel algorithm for balanced binary trees. *IEEE Trans. Software Eng.* **SE-12,** 3 (Mar. 1986), 442–449.
9. Wang, B. F., and Chen, G. H. Efficient parallel algorithms for constructing B-trees, in preparation.

BIING-FENG WANG was born in Taiwan, on January 15, 1965. He received the B.S. degree in computer science from National Chiao-Tung University, Taiwan, in June 1988. Since September 1989, he has been working toward a Ph.D. degree in computer science at National Taiwan University, Taiwan. His current research interests include design and analysis of algorithms, parallel and distributed computation, and computer networks.

GEN-HUEY CHEN was born in Taiwan, on October 10, 1959. He received the B.S. degree in computer science from National Taiwan University, Taiwan, in June 1981 and the M.S. and Ph.D. degrees in computer science from National Tsing Hua University in June 1983 and January 1987, respectively. In February 1987, he joined the faculty of National Taiwan University and he is now an associate professor in the Department of Computer Science and Information Engineering. His current research interests include design and analysis of algorithms, distributed algorithms, parallel computation, and parallel computer architectures.