An Efficient Parallel Algorithm for the Single Function Coarsest Partition Problem on the EREW PRAM

Kyeoung-Ju Ha[a], Kyo-Min Ku, Hae-Kyeong Park, Young-Kook Kim, and Kwan-Woo Ryu

In this paper, we derive an efficient parallel algorithm to solve the single function coarsest partition problem. This algorithm runs in $O(log^2 n)$ time using $O(nlogn)$ operations on the EREW PRAM with $O(n)$ memory cells used. Compared with the previous PRAM algorithms that consume $O(n^{1+\varepsilon})$ memory cells for some positive constant $\varepsilon > 0$, our algorithm consumes less memory cells without increasing the total number of operations.

Manuscript received February 16, 1998; revised March 23, 1999.
This work was supported by GRANT No. KOSEF 951-0100-001-2 from the Korea Science and Engineering Foundation.
[a]Electronic mail: kjha@etri.re.kr

I. INTRODUCTION

The single function coarsest partition problem can be described as follows. Given a set S of n elements, an initial partition $B = \{B_1, B_2, \cdots, B_k\}$ of S, and a function f on S, the problem is to form a new partition $Q = \{Q_1, Q_2, \cdots, Q_m\}$ in which each set $Q_i \in Q$ is a subset of some set $B_j \in B$, and each image set $f[Q_i]$ is a subset of some set $Q_l \in Q$. Q is the coarsest such a partition (i.e., Q has the fewest number of sets that satisfy the above constraints).

There are two well-known sequential algorithms to solve this problem. An $O(nlogn)$ time algorithm is given in [1], and a linear time algorithm appeared later in [2]. Several parallel algorithms have also appeared in the literature. In [3], JáJá and Kosaraju provide an $O(\sqrt{n})$ time algorithm on a $\sqrt{n} \times \sqrt{n}$ mesh of processors. Srikant describes an $O(log^2 n)$ time algorithm that uses $O(nlog^2 n)$ operations on the CREW PRAM [4]; Galley and Iliopoulos describe an $O(logn)$ time algorithm that uses $O(nlogn)$ operations on the Arbitrary CRCW PRAM [5]; Cho and Huynh provide an $O(logn)$ time algorithm that requires $O(n^3)$ operations on the EREW PRAM and $O(n^2)$ operations on the CREW PRAM [6]. Recently, JáJá and Ryu provide an $O(logn)$ time algorithm that requires $O(nloglogn)$ operations on the Arbitrary CRCW PRAM [7]. Note that these parallel algorithms except in [4] use $O(n^{1+\varepsilon})$ memory cells, where ε is a positive constant.

In this paper, we present a parallel algorithm that solves the single function coarsest partition problem in $O(log^2 n)$ time using $O(nlog n)$ operations on the EREW PRAM. The algorithm uses only $O(n)$ memory cells.

The rest of this paper is organized as follows. In Section **II**, the PRAM model is reviewed briefly and some well known results on the model are described. The overall strategy of our algorithm is explained in Section **III**. The special case when the graph induced by function f consists of a set of cycles is handled in Sections **IV** and **V**. The tree nodes and some remaining details are covered in Section **VI**. In Section **VII**, some concluding remarks are presented.

II. PRELIMINARIES

The model of parallel computation used in this paper is the EREW (Exclusive-Read Exclusive-Write) PRAM (Parallel Random Access Machine). The PRAM consists of p synchronous processors, $P_0, P_1, \cdots, P_{p-1}$, all having access to and exchanging data through a large shared memory. An EREW PRAM does not allow simultaneous access by more than one processor to the same memory location. The detail of this model is referred to [8], [9].

Given a sequence of n elements $(x_1, x_2, \cdots, x_n)$ and an associative operator $+$, the prefix sum problem is to compute the n partial sums defined by $s_i = x_1 + x_2 + \cdots + x_i,\ 1 \le i \le n$. The optimal algorithm for solving this problem is given in the following lemma.

Lemma 2.1. [10] *The prefix sums of a sequence of n elements can be computed in $O(log n)$ time using $O(n)$ operations on the EREW PRAM.* □

Sorting a list of n elements can also be performed optimally as follows.

Lemma 2.2. [11] *Given a list of n elements drawn from a linearly ordered set, the list can be sorted in $O(log n)$ time using $O(nlog n)$ operations on the EREW PRAM.* □

Given a linked list, the list ranking problem is to compute the distance of each node to the end of the list.

Lemma 2.3. [12] *Given a linked list of n nodes, the list ranking problem can be solved in $O(log n)$ time using $O(n)$ operations on the EREW PRAM.* □

III. THE OVERALL STRATEGY

Given a set S of n elements, an initial partition $B = \{B_1, B_2, \cdots, B_k\}$ of S, and a function f on S, we seek a new partition $Q = \{Q_1, Q_2, \cdots, Q_m\}$ of S that satisfies the following conditions:

1. Each set $Q_i \in Q$ is a subset of some set $B_j \in B$.
2. Each image set $f[Q_i] = \{f(x) \mid x \in Q_i\}$ is a subset of some set $Q_l \in Q$.
3. Q is the coarsest partition, i.e., Q has the fewest number of sets that satisfy the above two conditions.

Without loss of generality, we assume that $S = \{1, 2, \cdots, n\}$. Hence the input can be specified by two arrays $A_f[1..n]$ and $A_B[1..n]$ of size n respectively such that $A_f[x] = f(x)$, and $A_B[x] = A_B[y]$ if and only if both x and y are in the same set of B, for all x and y in S. We expect to determine the output as an array $A_Q[1..n]$ of size n such that $A_Q[x] = A_Q[y]$ if and only if both x and y are in the same set of Q. Thus, the single function coarsest partition problem can be regarded as a labeling problem which labels each element of S according to the final partition Q (Q-labeling), given the function f and the initial partition B (B-labels). Let $f^0(x) = x$ and $f^i(x) = f(f^{i-1}(x))$ for $i > 0$. The following simple lemma from [2] is helpful in motivating our solution.

Lemma 3.1.

(*i*) $\forall x, y \in S, A_Q[x] = A_Q[y]$ *if and only if* $A_B[x] = A_B[y]$ and $A_Q[f(x)] = A_Q[f(y)]$.

(*ii*) $\forall x, y \in S, A_Q[x] = A_Q[y]$ *if and only if* $A_B[f^i(x)] = A_B[f^i(y)], i = 0, 1, \cdots, n$. □

We can translate this problem into the following graph problem (cf. [3]). Construct a directed graph $G = (V, E)$ such that $V = S = \{1, 2, \cdots, n\}$ and $(x, f(x)) \in E$, $\forall x \in V$. Each node x is B-labeled, i.e., assigned the label $A_B[x]$. Our objective is to relabel each node such that any two nodes x and y are assigned the same Q-label if and only if both x and y are in the same set of Q.

Since the outdegree of each vertex in G is one, the graph G is a pseudo-forest. Each component of G is a pseudo-tree in which there is exactly one cycle and all the paths end in the cycle. Clearly, statements (i) and (ii) of Lemma 3.1 can be expressed as follows:

(*i*) Any two nodes x and y in V have the same Q-label if and only if x and y have the same B-label, and the parents of x and y have the same Q-label.

(*ii*) Let $x = (x_0, x_1, \cdots, x_n)$ and $y = (y_0, y_1, \cdots, y_n)$ be two directed paths of length n starting from x and y respectively. Note that $x_i = f^i(x)$ and $y_i = f^i(y)$, $i = 0, 1, \cdots, n$. Then, nodes x and y have the same Q-label if and only if nodes x_i and y_i have the same B-label, where $i = 0, 1, \cdots, n$.

Example 3.1. Given a function f and a partition B represented by the arrays $A_f[1..16] = [2,4,6,8,10,12,1,3,5,7,9,11,14,15,16,13]$ and $A_B[1..16] = [1,2,1,1,2,2,3,3,1,1,3,1,1,2,1,3]$. Then, $B = \{B_1, B_2, B_3\}$ and $B_1 = \{1,3,4,9,10,12,13,15\}$, $B_2 = \{2,5,6,14\}$ and $B_3 = \{7,8,11,16\}$. The corresponding digraph is shown in Fig. 1. Note that it consists of two simple cycles. The B-label of a node is given just outside of the circle. Note that nodes 1, 3, 9 and 13 will have the same Q-label, and nodes 1 and 4 will not have the same Q-label. □

Determining the Q-labels of all the nodes in G can be done by implementing the following strategy on the directed graph.

[Step 1] Mark all the cycle nodes in the pseudo-forests.
[Step 2] Assign the Q-labels to the cycle nodes.
[Step 3] Assign the Q-labels to the remaining tree nodes.

We will explain the implementations of these steps in the next three sections respectively.

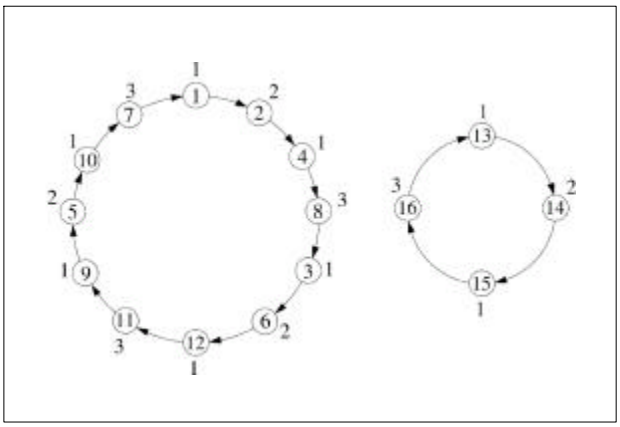


Fig. 1. The digraph corresponding to the instance given in Example 3.1.

IV. FINDING CYCLE NODES

Recall that the input consists of the two arrays $A_f[1..n]$ and $A_B[1..n]$ representing the function f and the B-labels respectively, and these two arrays can be interpreted as a directed graph whose nodes have been assigned the B-labels. The following algorithm identifies all the nodes in a cycle of G.

Algorithm 4.1: Finding Cycle Nodes

Input: *Two arrays* $A_f[1..n]$ *and* $A_B[1..n]$, *and* G.
Output: *All the cycle nodes are marked.*
Step 1: *For each edge* $(x, f(x))$, *create its buddy* $(f(x), x)$.
Step 2: *Construct an adjacency list of the modified graph and find the Euler tours in the pseudo-forest by using the procedure in [8], [13]. Then each pseudo-tree has two Euler tours. Now, a close observation of the resulting tours as determined by the successor function of [8], [13] indicates that there are two Euler cycles for each pseudo-tree, and each cycle edge* $(x, f(x))$ *and its buddy* $(f(x), x)$ *appear in different Euler cycles, while each tree edge* $(y, f(y))$ *and its buddy* $(f(y), y)$ *appear in the same Euler cycle. See Example 4.1.*
Step 3: *Determine and mark the nodes on the cycles.*

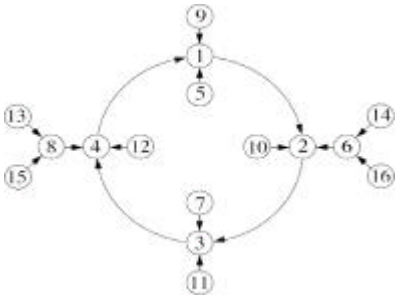
The correctness of Algorithm 4.1 follows the statements in Step 2 of the algorithm. Step 2 can be done by sorting the edges to construct an appropriate adjacency list; the corresponding Euler tours can be constructed easily from that adjacency list. Hence, Step 2 can be done in $O(\log n)$ time using $O(n \log n)$ operations. Step 3 can be done by Lemma 2.3 in $O(\log n)$ time using $O(n)$ operations. Hence, we have the following lemma.

Lemma 4.1. *Given two arrays* $A_f[1..n]$ *and* $A_B[1..n]$, *Algorithm 4.1 correctly marks all the cycle nodes in* $O(\log n)$ *time using* $O(n \log n)$ *operations on the EREW PRAM.* □

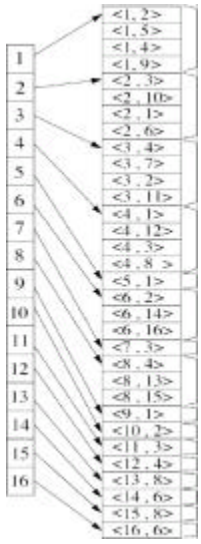
Example 4.1. Given a function f represented by the arrays $A_f[1..16] = [2,3,4,1,1,2,3,4,1,2,3,4,8,6,8,6]$. The corresponding digraph G is shown in Fig. 2(b). The modified digraph by Algorithm 4.1(Step 1) is Fig. 2(c). An adjacency list of the modified digraph G' is Fig. 2(d). Euler tours from the adjacency list is Fig. 2(e). The cycle nodes

A_f	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	2	3	4	1	1	2	3	4	1	2	3	4	8	6	8	6

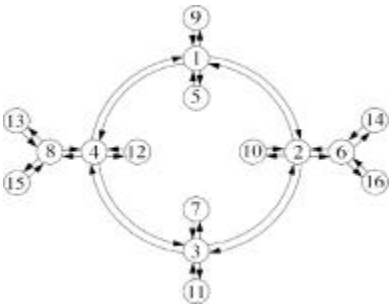
(a) Input array.



(b) The digraph G corresponding to Af.



(d) Adjacency list.



cycle cdge :
{<1,2>,<2,1>,<2,3>,<3,2>,<3,4>,<4,3>,<4,1>,<1,4>}
tree edge :
{<1,9>,<9,1>,<1,5>,<5,1>,<2,10>,<10,2>,<2,6>,<6,2>,
<6,14>,<14,6>,<6,16>,<16,6>,<3,7>,<7,3>,<3,11>,
<11,3>,<4,12>,<12,4>,<8,4>,<4,8>,<8,13>,<13,8>,
<8,15>,<15,8>}

(c) The modified digraph G' by Algorithm 4.1 (Step 1).

[E1] :
{<1,5>,<5,1>,<1,4>,<4,12>,<12,4>,<4,3>,<3,7>,
<7,3>,<3,2>,<2,10>,<10,2>,<2,1>,<1,5>,...}
[E2] :
{<1,9>,<9,1>,<1,2>,<2,6>,<6,14>,<14,6>,<6,16>,
<16,6>,<6,2>,<2,3>,<3,11>,<11,3>,<3,4>,<4,8>,
<8,13>,<13,8>,<8,15>,<15,8>,<8,4>,<4,1>,<1,9>,...}

(e) Euler tours $E1$ and $E2$ generated from the representation (d).

Cycle nodes :
{1, 2, 3, 4}
Tree nodes :
{5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}

(f) Cycle nodes and tree nodes in the digraph G (Fig. 2(b)).

Fig. 2. Steps of cycle node detection.

can be identified from the Euler tours, using the property that a cycle edge and its reverse belong to different Euler tours while a tree edge and its reverse belong to the same Euler tour. For example, cycle edges $<1,2>$ and $<2,1>$ belong to Euler tours $E2$ and $E1$ respectively, and tree edges $<2,6>$ and $<6,2>$ belong to $E2$ alone. The cycle and tree nodes in the digraph G (Fig. 2(b)) are shown in Fig. 2(f). □

V. LABELING CYCLE NODES

In this section, we consider the coarsest partition problem for a function whose graph representation consists of a set of cycles. We begin with a few definitions. Given a string S and a positive integer i, S^i represents the string S concatenated with itself i times. The smallest repeating prefix of a string S is the shortest prefix P of S such that $P^j = S$, for some $j > 0$. Note that in this case P is a

period of S. If x is any node of a cycle C of length k, then C can be represented as the circular string $(x, f(x), f^2(x), \cdots, f^{k-1}(x))$, together with the B-label string $(A_B[x], A_B[f(x)], A_B[f^2(x)], \cdots, A_B[f^{k-1}(x)])$. Let P be the smallest repeating prefix of the B-label string of C. Consider the sets

$$C_i = \{f^j(x) \mid j = 0, \cdots, k-1 \text{ and } j = i \bmod |P|\}$$
$$\text{where, } i = 0, \cdots, |P|-1 .$$

Then, by Lemma 3.1 (*ii*), any two nodes x and y from the same set C_i have the same Q-label, since $A_B[f^l(x)] = A_B[f^l(y)]$, $l = 0, 1, \cdots, n$. Similarly, any two nodes from different such sets can not have the same Q-label. Thus, given any two nodes x and y in C, $A_Q[x] = A_Q[y]$ if and only if $x, y \in C_i$, for some i. If $|P| = |C|$, the B-label string of C is not repeating, and hence every node in C has a different Q-label.

Example 5.1. Given the function f and the partition B introduced in Example 3.1, the corresponding graph has two cycles C and D. Cycle C and its B-label string are given by $(1, 2, 4, 8, 3, 6, 12, 11, 9, 5, 10, 7)$ and $(1, 2, 1, 3, 1, 2, 1, 3, 1, 2, 1, 3)$ respectively. Hence, the smallest repeating prefix P of the B-label string is $(1, 2, 1, 3)$, and $C_0 = \{1, 3, 9\}$, $C_1 = \{2, 6, 5\}$, $C_2 = \{4, 12, 10\}$, $C_3 = \{8, 11, 7\}$. Cycle D and its B-label string are given by $(13, 14, 15, 16)$ and $(1, 2, 1, 3)$ respectively, and hence $D_0 = \{13\}$, $D_1 = \{14\}$, $D_2 = \{15\}$, $D_3 = \{16\}$. Note that the nodes in $C_i \cup D_i$, $i = 0, 1, 2, 3$, have the same Q-label. If we set $Q_{i+1} = C_i \cup D_i$ for $i = 0, 1, 2, 3$, the output is given by $A_Q[1..16] = [1, 2, 1, 3, 2, 2, 4, 4, 1, 3, 4, 3, 1, 2, 3, 4]$. □

Given two distinct cycles C and D, let B_C and B_D be their corresponding B-label strings, and let P_C and P_D be the smallest repeating prefixes of B_C and B_D respectively. We say that P_C and P_D are cyclic shift equivalent (or $P_C \equiv P_D$) if and only if one is the cyclic shift of the other. We also define two cycles C and D to be equivalent if and only if $P_C \equiv P_D$. Note that C and D need not have the same length, even if C and D are equivalent. For example, cycles C and D of Example 5.1 are equivalent.

Let x and y be a pair of nodes such that $x \in C$ and $y \in D$, where C and D are equivalent, and let $|P_C| = |P_D| = l$. Assume that $P_C = (A_B[x], A_B[f(x)], \cdots, A_B[f^{l-1}(x)])$ and $P_D = (A_B[y], A_B[f(y)], \cdots, A_B[f^{l-1}(y)])$ and $A_B[f^i(x)] = A_B[f^i(y)]$, $i = 0, 1, \cdots, l-1$. Clearly, this can be achieved by shifting P_C or P_D cyclically whenever C and D are equivalent. Then, $f^i(x)$ and $f^i(y)$ must have the same Q-label, $i = 0, 1, \cdots, l-1$. Moreover, if we let

$$C_i = \{f^j(x) \mid j = 0, \cdots, |C|-1 \text{ and } j = i \bmod l\},$$
$$\text{where, } i = 0, \cdots, l-1,$$

and

$$D_i = \{f^j(y) \mid j = 0, \cdots, |D|-1 \text{ and } j = i \bmod l\},$$
$$\text{where, } i = 0, \cdots, l-1,$$

then all the nodes in $C_i \cup D_i$ have the same Q-label. That is, $\forall x, y \in C \cup D$, $A_Q[x] = A_Q[y]$ if and only if both x and y are in $C_i \cup D_i$, for some i.

Now, we describe an algorithm for solving the single function coarsest partition problem when the input consists of a set of cycles.

Algorithm 5.1: Cycle Node Labeling

Input: *Two arrays $A_f[1..n]$ and $A_B[1..n]$ representing the input function f and the initial partition B respectively. The graph representation of f consists of a set of cycles.*

Output: *An array $A_Q[1..n]$ such that $A_Q[x] = A_Q[y]$ if and only if both x and y have the same Q-label.*

Step 1: *Rearrange the input arrays Af and AB such that each cycle $(x, f(x), \cdots, f^{k-1}(x))$ and its B-label string $(A_B[x], A_B[f(x)], \cdots, A_B[f^{k-1}(x)])$ occupy consecutive memory locations.*

Step 2: *Partition the input cycles according to the cyclic shift equivalence relation defined above, and assign the appropriate Q-labels as above.*

The correctness of Algorithm 5.1 follows the discussion preceding the introduction of the algorithm. We now consider the implementation of the algorithm. Step 1 can be implemented as follows. First, we label each cycle with one of the indices of the cycle, and then rank all the nodes in the cycle starting from the chosen index. We can do this by using the list ranking which runs in $O(\log n)$ time using $O(n)$ operations. Once this information is available, we rearrange the input arrays A_f and A_B so that each cycle and its B-label string occupy consecutive memory locations according to the cyclic ordering. Hence, Step 1 can be done in $O(\log n)$ time using $O(n)$ operations.

Step 2 can be divided into two substeps. In the first substep, we find the smallest repeating prefix of the B-label string of each cycle that can be done in $O(\log n)$ time using $O(n)$ operations on the EREW PRAM [14]. In the second substep, we partition the cycles into equivalence classes and deduce the Q-label of the nodes. This substep can be done by first computing a minimal starting point for each cycle and then by sorting the B-label strings in $O(\log^2 n)$ time using $O(n \log n)$ operations. Please refer to [15] for detailed information. Hence, we have the following lemma.

Lemma 5.1. *The single function coarsest partition problem can be solved in $O(\log^2 n)$ time using $O(n \log n)$ operations on the EREW PRAM if the graph representation of the function is a set of cycles.* □

VI. LABELING TREE NODES

Let $G=(V,E)$ be the directed graph corresponding to an instance of the single function coarsest partition problem. Assume that all the cycle nodes have already been Q-labeled. In this section, we describe how to Q-label the remaining unlabeled nodes in G. The unlabeled nodes can be classified into two types; type one consists of the nodes having the same Q-labels as the cycle nodes, and type two consists of the remaining nodes. The following lemma is important to Q-label type one tree nodes. We assume that each tree T has been rooted at an arbitrary node of the cycle.

Lemma 6.1. *Let $T \subset G$ be a tree whose root r belongs to the cycle $C = (r = f^0(r), f(r), \cdots, f^{k-1}(r))$ of length k. Let x be any node at level l in T, where the level of r is zero. Then, $A_Q[x] = A_Q[f^{((k-(l \bmod k)) \bmod k)}(r)]$ if and only if $A_B[f^j(x)] = A_B[f^{((k-(l \bmod k)+j) \bmod k)}(r)]$, $j = 0, \cdots, l$. In other words, x has the same Q-label as one of the cycle nodes of its pseudo-tree if and only if each node in the path from x to r has the same B-label as its corresponding node in the cycle.*

Proof: The proof follows Lemma 3.1 (*ii*). See Example 6.1. □

Example 6.1. Given a function f and a partition B represented by the arrays $A_f[1..13] = [2,3,4,5,6,7,8,1,1,9,9,11,12]$ and $A_B[1..13] = [1,2,1,3,1,2,1,3,3,1,1,3,1]$. Then $B = \{B_1, B_2, B_3\}$, and $B_1 = \{1,3,5,7,10,11,13\}$, $B_2 = \{2,6\}$, and $B_3 = \{4,8,9,12\}$. The corresponding digraph is shown in Fig. 3. The B-label of a node is given just outside the circle representing the node. Note that node 4, 8, 9 will have the same Q-label by lemma 3.1 (ii). Nodes 5 and 13 can not have the same Q-label since the B-labels of their parents are different. □

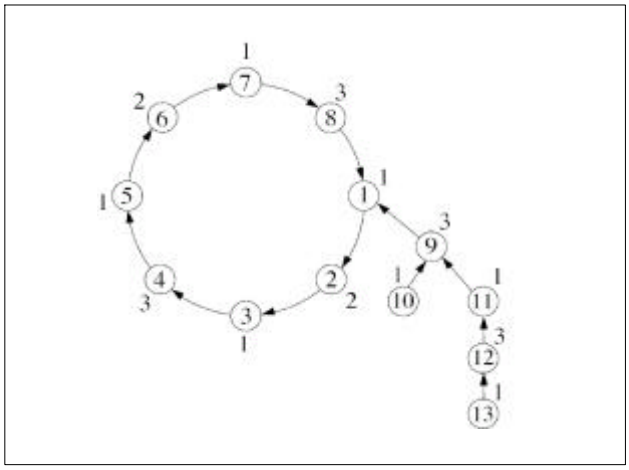


Fig. 3. The digraph corresponding to the instance given in Example 6.1.

Lemma 6.1 implies that if x is a tree node at level l such that $A_B[x] \neq A_B[f^{((k-(l \bmod k)) \bmod k)}(r)]$, then no descendant node of x has a Q-label that appears in any of the cycles in G. Below is our algorithm to Q-label type 1 tree nodes.

Algorithm 6.1: Type 1 Tree Node Labeling

Input: *A pseudo-forest G = (V, E). All nodes in the cycles of G have been Q-labeled and stored in consecutive memory locations. Each tree has been rooted at a node of its cycle.*

Output: *The Q-labels of type 1 tree nodes, and a forest F' consisting of type 2 tree nodes.*

Step 1: *For each tree node x, compute its level.*

Step 2: *Each tree node reads the B-label and the Q-label of its corresponding node in the cycle (Lemma 6.1), and compares its B-label with that of the cycle node. Mark x if they are the same.*

Step 3: *For each unmarked node, unmark all of its descendants. Note that, after this step, all the marked nodes are type 1 tree nodes, and the remaining tree nodes are type 2 tree nodes.*

Step 4: *Q-label all the marked nodes with the Q-labels of their corresponding nodes in the cycles.*

Step 5: *Let F be the forest consisting of type 2 tree nodes. Partition F into sets of trees $\{S_1, S_2, \cdots, S_k\}$ such that the parents of the roots of the trees in each set S_i have the same Q-label, $i = 1, 2, \cdots, k$. Construct a new forest F' by combining the trees of S_i into a single tree T_i. The root of T_i is a dummy node with the roots of the trees in S_i as its children. Note that any two nodes from different such trees can not have the same Q-label.*

Lemma 6.2. *Algorithm 6.1 correctly finds the Q-labels of all type 1 tree nodes and constructs the forest F' with type 2 tree nodes, it runs in $O(\log n)$ time using $O(n \log n)$ operations on the EREW PRAM.*

Proof: The correctness of Algorithm 6.1 is clear by Lemma 6.1. Steps 1 and 3 can be easily done by using the Euler tour technique in $O(\log n)$ time using $O(n \log n)$ operations. Step 4 is trivial to do. To handle the read conflicts in Step 2, we can use the sorting algorithm in Lemma 2.2. Step 5 can also be done by using the sorting algorithm, the lemma then follows. □

Now, we Q-label type 2 tree nodes. We can consider each tree in F' separately since any two nodes from different trees in F' cannot have the same Q-label. Let x and y be any two nodes in a tree T of F'. Then, we can easily show that x and y have the same Q-label if and only if the level l of x is the same as that of y in T and $f^i(x) = f^i(y)$, for all $i = 0, 1, \cdots, l-1$. The following algorithm computes the Q-labels of type 2 tree nodes by utilizing this property.

Algorithm 6.2: Type 2 Tree Node Labeling

Input: *Tree T with n nodes represented in $A_f[1..n]$ and $A_B[1..n]$. We assume that the B-label and the Q-label of the root are unique values and the parent of the root is the root. We also assume that every node v knows its own level level(v).*

Output: *An array $A_Q[1..n]$ such that $A_Q[x] = A_Q[y]$ if and only if both x and y have the same Q-label.*

Step 1: *If the depth of the tree T is < 2, Q-label the nodes in level 1 in such a way that nodes with the same B-label should have the same Q-label. Return.*

Step 2: *Let N_E and N_O be the numbers of nodes in even levels and in odd levels respectively. If $depth(T) = 2$ or $N_E < N_O$ then select even levels. Select odd levels otherwise.*

Step 3: *Assign new B-labels to the nodes on the selected levels in such a way that nodes whose B-labels are the same and whose parents have the same B-label should have the same new B-label. For each selected node v, set $A_f[v]$ as $A_f[A_f[v]]$, and set $level(v)$ as $\lfloor level(v)/2 \rfloor$.*

Step 4: *Let T_S be the tree composed of the root and the selected nodes in Step 2. Perform Steps 1, 2, and 3 recursively to Q-label the nodes of T_S.*

Step 5: *Q-label the nodes that were not selected in Step 2 in such a way that nodes whose B-labels are the same and whose parents have the same Q-label should have the same Q-label.*

Lemma 6.3. *Algorithm 6.2 computes the Q-labels of all type 2 tree nodes correctly. The algorithm runs in $O(\log n)$ time using $O(n \log n)$ operations on the EREW PRAM.*

Proof: The correctness of Algorithm 6.2 is also obvious. Steps 1, 2, 3, and 5 can be done easily in $O(\log n)$ time using $O(n \log n)$ operations on our model by using Lemmas 2.1 and 2.2. When $depth(T) > 2$, the number of nodes in T_S is at most the half number of nodes in tree T. Hence the total execution time $T(n)$ and the total number of operations $W(n)$ can be described by the following recurrence relations,

$$T(n) = T(n/2) + O(\log n),$$
$$W(n) = W(n/2) + O(n \log n).$$

Clearly, $T(n) = O(\log^2 n)$ and $W(n) = O(n \log n)$. □

Also we show without difficulty that the memory needed for performing our algorithms is only $O(n)$. Hence, we have the following theorem.

Theorem 6.1. *The single function coarsest partition problem can be solved in $O(\log^2 n)$ time using $O(n \log n)$ operations*

on the EREW PRAM. The memory used for the algorithm is $O(n)$. □

VII. CONCLUSION

In this paper we devised an efficient parallel algorithm to solve the single function coarsest partition problem which runs in $O(log^2 n)$ time using $O(nlogn)$ operations on the EREW PRAM with only $O(n)$ memory cells. Compared with the previous PRAM algorithms that consume $O(n^{1+\varepsilon})$ memory cells for some constant $\varepsilon > 0$, our algorithm consumes less memory cells without increasing the total number of operations.

The multi-function coarsest partition problem may be a more interesting problem, since its efficient solution can be applied for regular language recognition, text editor construction and string matching, etc. However this problem is not easy to solve, hence there is no known efficient parallel algorithm for solving the problem yet. Any efficient parallel algorithm for the coarsest partition problem is meaningful and is worthy of a future research.

REFERENCES

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[2] R. Paige, R. E. Tarjan, and R. Bonic, "A Linear Time Solution to the Single Function Coarsest Partition Problem," *Theoretical Computer Science*, Vol. 40, 1985, pp. 67—84.

[3] J. JáJá and S. R. Kosaraju, "Parallel Algorithms and Planar Graph Isomorphism and Related Problems," *IEEE Trans. on Circuits and Systems*, Vol. 35, No. 3, 1988, pp. 304—311.

[4] Y. N. Srikant, "A Parallel Algorithm for the Minimization of Finite State Automata," *Intern. J. Comput. Math.*, Vol. 32, 1990, pp. 1—11.

[5] C. N. Galley and C. S. Iliopoulos, "A Simple O(*nlogn*) Cost Parallel Algorithm for the Single Function Coarsest Partition Problem," *Parallel Processing Letters*, 1994.

[6] S. Cho and D. T. Huynh, "The Parallel Complexity of Coarsest Set Partition Problems," *Information Processing Letters*, Vol. 42, 1992, pp. 89—94.

[7] J. JáJá and K.W. Ryu, "An Efficient Parallel Algorithm for the Single Function Coarsest Partition Problem," *Theoretical Computer Science*, Vol. 129, 1994, pp. 293—307.

[8] J. JáJá, *An Introduction to Parallel Algorithms*, Addison-Wesley, 1992.

[9] R. M. Karp and V. Ramachandran, "Parallel Algorithms for Shared-Memory Machines," J. Van. Leewen, Ed., *Handbook of Theoretical Computer Science, Vol. A: Algorithms and Complexity*, MIT Press, Cambridge, MASS, 1990.

[10] R. E. Lander and M. J. Fisher, "Parallel Prefix Computation," *J. ACM*, 27, 1980, pp. 831—838.

[11] R. Cole, "Parallel Merge Sort," *SIAM J. Computing*, 17(4), 1988, pp. 770—785.

[12] R. J. Anderson and G. L. Miller, "Deterministic Parallel List Ranking," *Algorithmica*, Vol. 6, No. 6, 1991, pp. 859—868.

[13] R. E. Tarjan and U. Vishkin, "An Efficient Parallel Biconnectivity Algorithms," *SIAM J. Computing*, 14(4), 1985, pp. 862—874.

[14] A. Czumaj *et al.*, "Work-Time-Optimal Parallel Algorithms for String Problems," *Proc. 27th ACM Symp. on Theory of Computing*, 1995, pp. 713—722.

[15] K. J. Ha and K. W. Ryu, "An Optimal-Work Parallel Algorithm for String Sorting on the EREW PRAM," *Journal of KISS(A): Computer Systems and Theory*, Vol. 23, No. 6, June 1996, pp. 563—572.

Kyeoung-Ju Ha received the B.S., M.S. and Ph. D. degrees in computer engineering from Kyungpook National University, Taegu, Korea, in 1991, 1993 and 1996, respectively. She joined ETRI in 1996, and she is currently working as a senior member of engineering staff in Information Security Technology Division. Her research interests are design and analysis of parallel algorithms for combinatorial problems, information and network security.

Kyo-Min Ku received his B.S. and M.S. degrees in computer engineering from Kyungpook National University in 1993 and 1995, respectively. From 1995 to 1999, he worked as a researcher at the Korea Research and Development Information Center (KORDIC). He is currently a full time instructor in Taegu National University of Education, Taegu, Korea. His research interests include parallel processing, security, and multimedia.

Hae-Kyeong Park received the B.S. degree in computer science from the Changwon National University, Changwon in 1990, and the M.S. and Ph.D. degree in computer engineering from the Kyungpook National University, Taegu in 1992 and 1997, respectively. In 1997, she joined ETRI as a Senior Member of Research Staff. She is presently in Internet Technology Department, Switching and Transmission Laboratory, ETRI. Her research interests are design and analysis of parallel algorithms for combinatorial problems and fast address lookup algorithm for IP routing.

Young-Kook Kim received the B.S. and M.S. degree in electrical engineering from Kyungpook National University, Taegu, Korea, in 1983 and 1987, respectively. He worked as a researcher at the Naval and Ship Research Institute. From 1987 to 1998, he worked for ETRI as a senior member of engineering staff in Information Security Technology Division. He has been established From 2 Information & Communications Corporation, Taejon, Korea, in 1998, where he is currently the president. His research interests are design and analysis of parallel processing, information data security and network computing.

Kwan-Woo Ryu received the B. S. degree in electrical engineering from Kyungpook National University, Taegu, Korea, in 1980, and the M.S. degree in computer science from the Korea Advanced Institute of Science and Technology, Seoul, in 1982. He received his Ph.D. degree from the Department of Computer Science at the University of Maryland, College Park, in 1990. Currently, he is an associate professor in the Department of Computer Engineering, Kyungpook National University, Korea. His main research interests are design and analysis of parallel algorithms for combinatorial problems, and computer graphics.