1986

An O(n log n) Cost Parallel Algorithm for the Single Function Coarsest Partition Problem

A. Apostolico

C. S. Iliopoulos

Robert Paige

Report Number:
86-640

AN $O(n \log n)$ COST PARALLEL ALGORITHM FOR THE SINGLE FUNCTION COARSEST PARTITION PROBLEM

A. Apostolico and C. S. Iliopoulos

Robert Paige
New York University

CSD-TR-640
November 1986

An $O(n \log n)$ Cost Parallel Algorithm for the Single Function Coarsest Partition Problem

by

A. Apostolico and C.S. Iliopoulos
Purdue University
Dept. of Computer Science
West Lafayette
IN 47907, U.S.A.

Robert Paige
New York University
Dept. of Computer Science
New York City
NY 10012

ABSTRACT

An algorithm on a CRCW PRAM for computing the coarsest refinement of a partition of a finite set S with respect to a function f over S is presented here. It requires $O(n)$ processors, $O(n \log n)$ space, and $O(\log n)$ time in the worst case.

1. Introduction

Here we study the complexity (on parallel computational models) of the "single-function coarsest partition" problem : given a finite set S, a partition $\pi = \{B_1, B_2, \cdots, B_k\}$ of S, and a function $f: S \rightarrow S$, compute the unique partition $\pi' = \{Q_1, Q_2, \cdots, Q_m\}$ of S that satisfies the following conditions:

(i) π' is a refinement of π; i.e.,

$$\forall\, Q \in \pi', \exists\, B \in \pi \mid Q \subseteq B$$

(ii) π' respects the function f ; that is,

$$\forall\, Q_1 \in \pi', \exists\, Q_2 \in \pi' \mid f[Q_1] \subseteq Q_2$$

(iii) π' is the coarsest partition satisfying conditions (i) and (ii); i.e., it has the fewest number of sets.

The computational model used here is CRCW PRAM (Concurrent Read-Concurrent Write Parallel RAM). The processors are unit-cost RAM's that can access a common memory. Some

processors can both read from and write into the same memory location. All operations involving different memory locations can also be done concurrently.

We measure the complexity of parallel algorithms by the pair (t, p), where t denotes the time and p the number of processors, and both are dependent on the size of the input. Furthermore, the product $t\ p$ is said to be the *cost* of the algorithm; the cost of an algorithm is essentially the running time of the algorithm with only one processor available.

In the literature several sequential algorithms for the single-function problem are cited. In [8], Paige and Tarjan give an $O(n)$ optimal algorithm for the one function partitioning problem improving the worst-case complexity bounds for the same problem given by Aho, Hopcroft and Ullman in [1]. Furthermore Hopcroft in [4] gives an $O(n \log n)$ algorithm for the many functions coarsest partition problem; i.e., the partition is required to respect a set of functions $f_1, \cdots, f_k$; Paige and Tarjan in [9] give an $O(m \log n)$ algorithm for the relational partitioning problem; i.e., the refinement should respect a binary relation with m pairs. In this paper we will only consider the single function coarsest partition problem.

In [5] an algorithm on a CRCW PRAM for computing the coarsest refinement of a partition of a set S with respect to a function f is presented; it requires $O(\log n)$ time and $O(n^2)$ processors. A modified version of this algorithm reduces the cost to $O(n^2)$, where the time is $O(n^x)$ for some $0 \le x \le 1$. Ja Ja and Kosaraju [7] recently claimed a PRAM solution that runs in $O(\log n)$ time with $O(n^2)$ processors and $O(n^2)$ space. In this paper we present an algorithm that requires $O(\log n)$ time, $O(n \log n)$ space, and $O(n)$ processors, which improves the bound given in [5].

2. An Outline Of the Algorithm

We refer to the sets belonging to a partition as blocks. The B-label of an element x in S is said to be the index of the block containing x in the initial partition π. The Q-label of an element x in S is said to be the index of the block containing x in the final partition π'.

Our algorithm makes use of the following lists (stored in the common memory of the CRCW PRAM)

(i) L_B : $L_B(x)$ is the B-label of an element x in S.

(ii) L_Q : $L_Q(x)$ is the Q-label of an element x in S.

Our algorithm exploits certain properties in the structure of the input function f. In particular, the graph of f can be characterized as follows:

(i) Each connected component contains only one cycle with zero or more trees leading into and rooted in the cycle.

(ii) Each tree can be partitioned into two parts - an "initial segment" and a "final segment". A nonroot tree node x (leading into a cycle C) is said to be in the *initial segment* if and only if either $f(x)$ is the root r or is in the initial segment, and x has the same B-label as $f^{-l}(r) \cap C$, where l is the length (i.e., number of edges) of the path from x to r; otherwise x is said to be a member of the *final segment.*

After analyzing the structure of function f, our algorithm determines the coarsest partition by computing Q-labels based on the following theorem.

Theorem 2.1 (*Paige* and *Tarjan* [8])

Let $f : S \rightarrow S$ be a finite function represented by its graph $G = (S,E)$ with nodes S and edges $E = \{ (s, f(s)): s \in S \}$. Let π be the initial partition of S (represented by B-labels). The Q-labels (representing the coarsest partition π') of the elements of S satisfy the following:

(i) Let C be a cycle of k elements and let P be the smallest repeating prefix of the string $[L_B(x), L_B(f(x)), L_B(f^2(x)), \cdots, L_B(f^{k-1}(x))]$. Then the elements of C satisfy

$$L_Q(f^j(x)) = L_Q(f^{i|P|+j}(x)), \quad 0 \le j < |P|, \quad 1 \le i \le (k / |P|) - 1$$

where $|P|$ is the length of P.

(ii) Two elements x and y of S that belong to two different cycles C_1 and C_2 satisfy

$$L_Q(f^j(x)) = L_Q(f^j(y)), \quad 0 \le j < |P|$$

if and only if P is the smallest repeating prefix of both strings

$[L_B(x), L_B(f(x)), \cdots, L_B(f^{k_1-1}(x))]$ and $[L_B(y), L_B(f(y)), \cdots, L_B(f^{k_2-1}(x))]$,

where k_1, k_2 are the lengths of C_1 and C_2 respectively.

(iii) The elements of the "initial segment" of the tree leading to a cycle C satisfy

$$L_Q(u) = L_Q(f^{-l}(r) \cap C)$$

where u is a tree node of the initial segment, r is the root of that tree and l is the length of the path from u to the root.

(iv) The elements of the "final segment" satisfy

$$L_Q(x) = L_Q(y)$$

if and only if

$$L_B(x) = L_B(y) \text{ and } L_Q(f(x)) = L_Q(f(y))$$

Based on the preceding logic, the algorithm proceeds in the following way.

(1) Find the set of cycles;

(2) Determine Q-labels for the cycle elements by lexicographically sorting the smallest repeating prefixes of the least lexicographic representation of the B-labels of each cycle.

(3) Determine Q-labels for the tree elements of the initial segments.

(4) Determine Q-labels for the tree elements of the final segments.

3. Finding the Set of Cycles and Labeling the Cycle Elements

All the cycles can be found in $O(logn)$ time with $O(n)$ processors and $O(n)$ space on a CRCW PRAM using Tarjan and Vishkin's biconnected component algorithm [11]. The computational steps for labeling the elements of the cycles are as follows:

(i) Compute the least lexicographic representation (abbreviated l.l.r) of the circular string $(L_B(x), L_B(f(x)), \cdots, L_B(f^{|C|-1}(x)))$ for each cycle C. An $O(n)$ sequential algorithm for the l.l.r problem was given in [10].

(ii) Compute the smallest repeating prefix (abbreviated s.r.p.) of every l.l.r computed in (i), using Galil's string matching algorithm, given in [3].

(iii) Assign Q-labels to every element of the cycles, according to the rules (i) and (ii) of Theorem 2.1

Lemma 3.1

There exists an algorithm on a CRCW PRAM that finds the l.l.r of a circular string $a_1,\ldots,a_n$ over the alphabet $\{1,\ldots,n\}$ in $O(\log n)$ time with n processors.

Proof(Sketch)

Below we describe the algorithm using a stronger version of the CRCW PRAM; when two processors attempt to write in the same memory location the one with the smallest index succeeds. In our application we can show that this form of PRAM is equivalent to the CRCW PRAM with non-deterministic solution of concurrent writes conflicts. Equivalence is shown by using the simulation of [12] (see also [3]), but in general the equivalence of the two models is an open problem.

W.l.o.g we assume that n is a power of 2. The algorithm is as follows:

(i) Processor p_i is assigned to a_i for $1 \le i \le n$. We make use of an $1 \times n$ array AUX (initialized to 0). In the final AUX, if $AUX(i) = 1$, then a_i is not a starting point of the l.l.r and if $AUX(i) = 0$ then a_i is a starting point of the l.l.r

(ii)For $k = 0, \ldots, \log n - 1$ do the following: Compare (lexicographically) the strings $x = (a_{q2^{k+1}+1}, \ldots, a_{(1+2q)2^k})$ and $y = (a_{1+(1+2q)2^k},\ldots,a_{(1+q)2^{k+1}})$ with $q = 0,\ldots,n/2^{k+1}-1$. If $x \ge y$, then let $AUX(q2^{k+1}+1) = 1$; if $x < y$ then, let $AUX(1+(1+2q)2^k) = 1$. The

comparison of the two strings requires constant time.

The correctness of the algorithm is straightforward. The time complexity follows immediately from the preceding discussion and the fact that the simulation of [12] requires constant time.□

Theorem 3.1

Suppose that f is a permutation on n points. Then there exists an algorithm on a CRCW PRAM for computing the coarsest refinement of a partition π with respect to the function f in $O(\log n)$ time, $O(n)$ space, and n processors.

Proof

We analyze the method above by examining cases (i)-(iii).

(i) Lemma 3.1.

(ii) Let α be a l.l.r. of a cycle and let β be the string $\alpha\alpha$ with its first character deleted. Using Galil's algorithm in [3] we find the first occurance of α in β; let $\beta = x\,\alpha y$. It is not difficult to see that x is the smallest repeating prefix. This can be done (see [3]) in $O(\log|\alpha|)$ time using $|\alpha|$ processors. Therefore the computation of all s.r.p.'s can be done in $O(\log n)$ time using n processors.

(iii) In order to assign the Q-labels to the cycle elements , one can see that it suffices to compute all the different equality classes represented by the s.r.p.'s. The computation of the classes can be done in $O(logn)$ time using n processors, as follows: The comparison of two strings can be done in constant time as in Lemma 3.1 with number of processors used equal to the sum of the lengths of the two strings. Therefore one can use the sorting network of Ajtai,M.,Komlos,J., Szemeredi,E., in [1] to sort the list of the strings, where each comparison of their algorithm is simulated as it was mentioned above. Furthermore one could separate the set of strings into subsets of strings of equal length and the use lexicographical sorting (see [6]) to sort them. This

method yields the same complexity bounds. □

4. Labeling the Tree Elements

This section deals with finding Q-labels for the tree elements. First we present a parallel procedure for computing the length of the path of each element to the root. Next, we give a procedure for labeling the initial segments. After that, we label the final segments.

4.1. Computing the path lengths

An algorithm for computing the label of a path is presented below: given a label $L(x) \in \{0, 1\}$ for each node x of a tree and an associative, commutative operation $\oplus$ over $\{0, 1\}$, the label of a path $(x_1 = x, x_2, \cdots, x_k = y)$ from x to y is said to be

$$L(x_1) \oplus L(x_2) \oplus \cdots \oplus L(x_k)$$

If addition is the $\oplus$ operation and $L(x) = 1$ for every x in the tree T, then the label of a path is merely the length of the path plus one.

Later, we will also need a procedure that splits a path $(x_1, \cdots, x_n)$ into two subpaths $(x_1, \cdots, x_k)$, $(x_{k+1}, \cdots, x_n)$, where x_{k+1} is the first node in the path that has a certain property. This can be done by means of computing the labels of the paths, using logical & as a $\oplus$-operation and labeling a node with 0 if the node has the given property or with 1 otherwise. This procedure, discussed in section 4.2, is used to separate the initial and final segments of trees.

Given a tree with a set of nodes T (excluding the root r) together with a labeling function L, and an operation $\oplus$, the following procedure computes the labels of all paths from every node to the root.

Procedure LABEL (T, $\oplus$, L, r)

begin

/* next $(x) \leftarrow$ the parent of x in T; */

for each node x in T **pardo**

$L_{\otimes}(x) \leftarrow L(x)$;

odpar

end

for each node x in T **pardo**

if next $(x) \neq r$ **then**

$L_{\otimes}(x) \leftarrow L_{\otimes}(x) \oplus L_{\otimes}(\text{next } (x))$;

next $(x) \leftarrow$ next (next (x));

odpar

end □

Proposition 4.1

The procedure above correctly computes the path-labels in $O(\log |T|)$ time, $O(|T|)$ space, and $O(|T|)$ processors, where $|T|$ denotes the number of nodes of T. □

4.2. Finding the elements of the initial segment and Assigning labels to them.

Now we shall refer to the labeling of the initial segment.

(1) For every tree and for every node of the tree compute in parallel the length l of the path to the root of the tree using the "continuous doubling" technique.

(2) If the B-label of an element x is not equal to the B-label of the l-th predecessor of the root, then remove the node x. This is done for all x's in parallel.

(3) Tree elements that remain connected to a cycle belong to the initial segments. The other tree elements belong to the final segments. The Q-label of an element x in an initial tree segment equals the Q-label of the l-th predecessor of the root of the tree, where l is the length of the path from x to the root.

The algorithm below splits a tree T into its initial segment and its final segment. This is achieved by invoking procedure LABEL for two purposes. Label is used to compute the path labels of every path from a tree element to the root. It is also used to determine elements of the initial segment, where the $\oplus$-operation is logical & and the node labels are defined in the following way:

$$L(x) = \begin{cases} 1 & L_B(x) = L_B(f^{-1}(r) \cap C) \\ 0 & \text{otherwise} \end{cases}$$

If the path label of a path from x to r is zero, then x belongs to the final segment; if the label is 1, then x belongs to initial segment.

Algorithm 4.2

INPUT: A tree T with root r, and a cycle C ($f^i(r)$, $1 \le i \le k$, are the elements of the cycle) together with the list L_Q defined on C.

OUTPUT: The elements of the initial segment together with their Q-labels

begin

for each element x in T **pardo**

$l \leftarrow L_+(x) \bmod k$;

if $L_B(x) = L_B(f^{-l}(r) \cap C)$ **then** $L(x) \leftarrow 1$;

else $L(x) \leftarrow 0$;

odpar

$L_{\&} \leftarrow$ LABEL(T, &, L)

for each element x in T **pardo**

if $L_{\&}(x) = 1$ **then** $L_Q(x) \leftarrow L_Q(f^{-l}(r) \cap C)$;

odpar □

Proposition 4.3

The algorithm above correctly finds the initial segment of the tree T and the Q-labels of its nodes in $O(\log |T|)$ time, $O(|T|)$ space, and $O(|T|)$ processors. □

4.3. Computing the labels of the final segment.

We can parallelize the labeling of the final segments as follows:

(1) Suppose that C denotes the subset of S that already has known Q-labels and the final segment F is attached to C. We partition the set F into blocks such that two elements of F belong to the same block if and only if the length of the path from both elements to the closest element of the initial segment is the same.

(2) Next we refine the above partition with respect to the B-labels, i.e., we split each block into sets with the same B-label.

(3) We further refine each partition π_i as follows; each block is split into blocks whose elements have f-images with the same Q-label. Although this computation seems to be of sequential nature, we make use of a "partial sum tree" to carry out this task.

The computational steps for labeling the final segment are as follows:

(i) Compute the sets

$$P_l := \{x: \text{there is a path of length } l \text{ from } x \text{ to } C\} \quad 1 \le l \le t.$$

(ii) Partition each P_l set into blocks $B_1^{(l)}, \cdots, B_{i_l}^{(l)}$, where $B_j^{(l)}$ contains all elements of P_l with the same B-label.

(iii) Compute a refinement $\pi_1 = \{A_1, \cdots, A_k\}$ of $\{B_1^{(l)}, \cdots, B_{i_l}^{(l)}\}$ such that

$$L_Q(f(x)) = L_Q(f(y)), \; x, y \in A_i \quad 1 \le i \le k$$

i.e., all elements of A_i, $1 \le i \le k$, have f-images with the same Q-label (Note that their image lie in C whose Q-labels are known). Moreover let

$$\pi_l = \{B_1^{(l)}, \ldots, B_{i_l}^{(l)}\}, \quad 2 \le l \le t$$

(iv) Each partition π_l is associated with a set

$$M_l = \{a \in \pi_l : f^{-1}(a) = \emptyset\}$$

i.e., the set of leaves of the final segment intersected with π_l.

(v) Recursively construct a balanced binary tree with leaves $\pi_1, \pi_2, \cdots, \pi_t$ as follows:

The parent of two partitions of π_l *and* π_{l+1} *is defined to be the coarsest refinement* $\pi_{l,l+1} = \{G_1, \cdots, G_m\}$ *of* π_{l+1} *such that*

$$f^y(G_i) \subseteq B_{j_i}^{(l)}, \quad \text{for some } j_i, 1 \le i \le m$$

where y *denotes the length from* $\pi_{l,l+1}$ *to the leaves.* i.e., *the f-image of* G_i, $1 \le i \le m$, *is a subset of a block of* π_l. *Also we associate the partition* $\pi_{l,l+1}$ *with the set* $M_{l,l+1} = M_{l+1}$.

(vi) Update the nodes of the tree, starting from the root, going towards the leaves as follows:

Replace the partition π_l (*child of* $\pi_{l,l+1}$) *by the coarsest refinement* $\pi_l' = \{E_1, E_2, \cdots, E_r\}$ *of* π_l *such that* E_i *is a subset of a block of the partition* π_l *whose preimage lies in a block of* $\pi_{l,l+1}$ *i.e.,*

$$E_i - (E_i \cap M_l) \subseteq f^y(G_{j_i}), \text{ for some } j_i \quad 1 \le i \le r$$

Also replace π_{l+1} *by its parent* $\pi_{l,l+1}$.

(vii) Let $S' = \bigcup_{l=1}^{t} M_l = S - f(S)$ .Then compute the coarsest refinement of π_l, for $1 \le l \le t$, in the following way :

Partition $E_i \cap M_l$ *for* $1 \le l \le t$ *and every block of each* π_l *into* $\{X_1, X_2, \cdots, X_m, Y\}$ *where*

$$L_Q(f^l(z) = L_Q(f^l(w)), \forall\, z, w \in X_i,\ 1 \le i \le m+1$$

and $X_{m+1} = D_i \cup Y$ (Note that Y could be empty or $m = 0$).

Theorem 4.4

Suppose that $\pi' = \bigcup_{l=1}^{t} \pi_l$ is the partition of the final segment computed above. Then

$$L_Q(f^l(a)) = L_Q(f^l(b)) \tag{4.1}$$

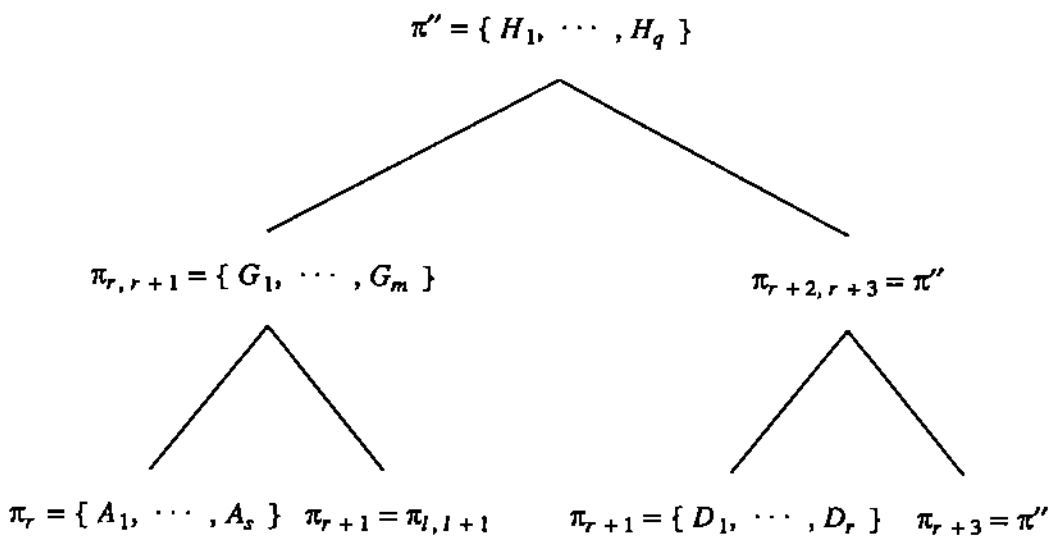
for any a,b members of a block of π_l, $1 \le l \le t$.

Proof

The proof is by induction of l. For $l = 1$, partition π_1 satisfies condition (4.1).

Now assert that (4.1) holds for $l \le r + 1$. We shall show that π_{l+2} satisfies (4.1).

Let us consider the following portion of the binary tree constructed above



Rule (v) above guarantees that

$$f^2(H_i) \subseteq G_{j_i}, \text{ for some } j_i,\ 1 \le i \le q \tag{4.2}$$

Also Rule (vi) implies that

$$D_i - (D_i \cap M_{r+2}) \subseteq f(H_{v_i}), \text{ for some } v_i,\ 1 \le i \le v \tag{4.3}$$

>From (4.2) and (4.3) it follows that

$$f^{r+2}(D_i - (D_i \cap M_{r+2}) \subseteq f^{r+3}(H_{v_i}) \subseteq f^{r+1}(G_{j_u})$$

which implies that (4.1) holds for $D_i - D_i \cap M_l$. Moreover Rule (vii) implies that $D_i \cap M_l$ satisfies (4.1) and this completes the proof.□

Theorem 4.5

There exists an algorithm for computing the Q-labels of the final segment in $O(\log n)$ time using $O(n)$ processors and $O(n \log n)$ space.□

5. REFERENCES

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

[2] Ajtai,M.,Kolmos,J.,Szemeredi,E., *An O(log n) sorting network* Combinatorica, Vol 5, (1) 1983,p 1-19

[3] Z. Galil, *Optimal parallel algorithms for string matching*, Proc. 16-th ACM Symp. on Theory of Computing, 1984, 240-248.

[4] Hopcroft, J.E., *An nlogn algorithm for minimizing states in a finite automation*, in: Kohavi and Paz, ed., Theory of Machines and Computations, Academic Press, NY, 1971, pp. 189–196.

[5] Iliopoulos, C.S., *A logarithmic time parallel algorithm for partitioning*, Purdue University, CSD-TR-603, (1986).

[6] Iliopoulos,C.S., *A log-time parallel algorithm for lexicographical ordering* Purdue University,CSD-TR-602,(1986)

[7] Ja'Ja' and Kosaraju, SIAM National Meeting, Boston, (1986)

[8] Paige, R., R. Tarjan, R. Bonic, *A linear time solution to the single function coarsest partition problem*, Theoretical Computer Science 40 (1985) 67–84.

[9] Paige, R., R. Tarjan, *Three efficient algorithms Based on Partition refinement*, to appear in SIAM J. Computing.

[10] Shiloach, Y., *Fast canonization of circular strings* J. Algorithms 2 (1981) 107-121

[11] Tarjan, R. and Vishkin, U., *An efficient parallel biconnectivity algorithm*, SIAM J. Comput. 14 (1985) 862-874.

[12] F. E. Fitch, R. L. Radge and A. Wigderson, *Relation between concurrent-write models of parallel computation, typescript*, Div of Comp. Science, Univ. of California at Berkeley (Nov. 1983).