1986

A Logarithmic Time Parallel Algorithm for Partitioning

Costas S. Iliopoulos

Report Number:
86-603

A LOGARITHMIC TIME PARALLEL ALGORITHM FOR PARTITIONING

Costas S. Iliopoulos

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

CSD TR #86-603
June 1986

A LOGARITHMIC TIME PARALLEL ALGORITHM FOR PARTITIONING

Costas S. Iliopoulos

Purdue University

Dept. of Computer Science

West Lafayette, IN 47907

U.S.A.

ABSTRACT

Here a parallel algorithm for computing the coarsest refinement of a partition of a set S with respect to a given function that requires $O(\log n)$ units of time and makes use of $O(n^2)$ processors, where n is the cardinality of the set S; the model of computation used, it is a Concurrent Read Exclusive Write Parallel RAM (abbreviated CREW PRAM). Furthermore an $O(n^2)$ cost algorithm for the same problem is presented.

1. Introduction

The classification of various computational problems into groups according to their computational complexity is one of the major tasks of Theoretical Computer Science. The class P of problems that are polynomial time solvable on sequential models (e.g. Turing machines, RAM) is generally agreed to be sequentially feasibly solvable problems. Here we are interested in the classification of problems according their computational complexity on parallel models of computation. A well known complexity class of efficiently solvable problems in parallel is NC (Nick's (Pippenger) Class), i.e., problems solvable by parallel algorithms in polylog time ($O(log^k n)$ for some constant k, with n the size of the input) with a polynomial number of pro-

cessors.

Here we show that the "single-function coarsest partition" problem is in the class NC. Given a set S of elements and a partition $\pi = \{A_1, A_2, \cdots, A_k\}$ of S, and a function $f: S \rightarrow S$, we want to compute a partition $\pi' = \{B_1, B_2, \cdots, B_m\}$ that satisfies the following conditions:

(i) The partition π' is a refinement of π, i.e.,

$$B_i \subseteq A_{j_i}, \forall\, i, \text{ for some } 1 \leq j_i \leq k$$

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

$$f(B_i) \subseteq B_{j_i}, \forall\, i \text{ for some } 1 \leq j_i \leq m$$

(iii) Partition π' is the coarsest one, satisfying conditions (i) and (ii).

In [5], 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 [2] yields 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 $f_1, \ldots f_k$; Paige and Tarjan in [6] gave an $O(n \log n)$ algorithm for the relational partitioning problem, i.e., the refinement should respect a binary relation. Here we shall deal only with the single function coarsest partition problem.

The computational model used here is CREW PRAM (Concurrent Read- Exclusive Write Parallel RAM). The processors are unit-cost RAM's that can access a common memory. Some processors can access the same memory location: they can concurrently read but they can not concurrently write. All operations involving different memory locations can be done concurrently.

Here we measure the complexity of parallel algorithms by the pair (t, p) where t denotes the time and p the number of processors, both dependent on the size of the input. Also the pro-

duct $t\ p$ is called the *cost* of the algorithm; the cost of an algorithm is essentially the running time of the algorithm with one only processor available and therefore can be used for finding the speeding up factor.

Here we present an algorithm for computing the coarsest refinement of a partition of a set S with respect to a function f, whose complexity is dominated by $O(log\ n)$ units of time and uses $O(n^2)$ processors. A modified version of this algorithm can be shown having a reduced cost of $O(n^2)$, in the expence of the running time, which in this case is $O(n^x)$ for some $0 \leq x \leq 1$.

Among the applications of the partitioning problem is the reduction of the number of states of finite automata, (see [2]), tree isomorplism (see [1] and [6]), graph isomorphism refinement and Congruence Closure (see [7]) and a surprising application to the automation of woven fabric on looms (see [5]).

2. An Outline of The Algorithm

Let $\pi = \{A_1, A_2, \cdots, A_k\}$ be the initial partition of S. A key operation in our algorithm is the partitioning with respect to f^{-m} for some integer m (see [2]); a refinement of π with respect to $f^{-m}(A_i)$, for $1 \leq i \leq k$ is defined to be

$$\pi' = \{B_{ij} := A_j \cap f^{-m}(A_i),\ 1 \leq i \leq k\ ,\ 1 \leq j \leq k\}, \tag{2.1}$$

After refining the initial partition with respect to f^{-m}, we associate each element of S with a pair (i, j) called *fingerprint* (denoted by $f_{s,m}$), where i is the index of the set that s belongs to in π and j is the index of the set that s belong to in π'.

The main step consists of computing refinements of the initial partition π with respect to f^{-m}, for $1 \leq m \leq n$ in parallel together with the corresponding fingerprints. After the implementation of the above n parallel steps, we have the sets of fingerprints $F_1, F_2, \cdots, F_n$. Based on these, it is determined if two or more elements belong to the same set at the final refinement. If these elements have the same fingerprints in every step, then they belong to the same set in final

partition. Sorting networks are used for computing the final partition by comparing the fingerprint sets.

The sequel is organized as follows: in section 3 we present an algorithm for the parallel computation of fingerprints, in sections 4 we give a parallel algorithms for computing powers of the given function f: $f(S), f^2(S), \cdots, f^n(S)$, in section 5 we discuss the fingerprint sorting and in section 6 we present the overall algorithm and we give a proof of its correctness and its running time analysis.

3. An Algorithm for splitting the Blocks

We recall that the "fingerprint" of an element s is an ordered pair of integers (i, j) such that the block B_{ij} of the partition π' (see (2.1)) contains s. We can view the blocks of the partition π' as

$$B_{ij} = \{ s : f^m(s) \in A_i \text{ and } s \in A_j \} \tag{3.1}$$

that it is equivalent to (2.1). At this point we are not focusing in computing the actual B_{ij}'s but we want to find out the fingerprints of each element of S. The formulation (3.1) is more helpful complexity -wise, since the size of the inverse image can be larger by a factor of $O(n)$ than the image of the function f.

The computation of the intersection makes use of "associate addressing". In the common memory we have a list L such that $L(s) = m$, where s belongs to A_m and s is an element of S

3.1. Lemma

There exists an algorithm on CREW PRAM that computes the above list L in constant time and makes use of $O(n)$ processors. □

Given the initial partition π and the function f^m, one can compute the fingerprints $f_{s,m}$ as follows:

3.2. Algorithm

begin

for each s in S **pardo**

$f_{s,m} \leftarrow (L(s) , L(f^m(s)))$

odpar

end □

3.3. Proposition

Algorithm 3.2 correctly computes the fingerprint with respect to f^m in constant time and it makes use $O(n)$ processors.□

4. Computing Powers Of The Function f

First we present an algorithm for computing powers of 2 of the function f, i.e., f^{2^i}, $1 \le i \le \lceil \log n \rceil$. This procedure will be used as a subroutine for the computation of all powers of the function f later.

4.1. Algorithm

INPUT: W.l.o.g we assume that $S = \{1, \cdots , n\}$ and a function $f : S \rightarrow S$.

OUTPUT: For each element s of S, $f^{2^i}(s)$ for $1 \le i \le \lceil \log n \rceil$.

begin

1. **for** $k = 1$ **to** $\lceil \log n \rceil$ **do**
2. **for each** s **in** S **pardo**
3. $e \leftarrow 2^{k-1}$;
4. $f^{2e}(s) \leftarrow f^e(f^e(s))$;

comment The values of $f^{2e}(s)$, for all s in S are stored in the common memory.

odpar

od

end. □

4.2. Theorem

Algorithm 4.1 correctly computes the required powers of 2 of the function f in $O(log\ n)$ units of time and makes use of $O(n)$ processors.□

In order to compute $f^k(S)$, we make use the above algorithm for computing powers of 2 of the function f , together with the binary expansion of k as follows:

4.3. Algorithm

INPUT: A set $S = \{1, \cdots, n\}$ and a function $f : S \to S$.

OUTPUT: A table for $f^k(S)$, for $1 \le k \le n$, $\forall\ s \in S$.

begin

1. compute $f^{2^i}(s)$, for $1 \le i \le \lceil \log n \rceil$, $\forall\ s \in S$ using algorithm 4.1;
2. **for** each s in S **pardo**
3. **for** $k = 1$ **to** n **with step** $\lceil \log n \rceil$ **pardo**
4. $f^k(s) \leftarrow f^{2^{r_0} + 2^{r_1} + \cdots + 2^{r_t}}(s)$;

   comment We make use of the binary expansion of k.
5. **for** $j = 1$ **to** $\lceil \log n \rceil$ **do**
6. $f^{j+k}(s) \leftarrow f(f^{j+k-1}(s))$;
7. **od**
8. **odpar**
9. **odpar**

end.□

4.4. Theorem

Algorithm 5.1 correctly computes $f^k(s)$, $1 \leq k \leq n$, for each element of S in $O(log\ n)$ units of time and makes use of $O(n^2/\log n)$ processors.

Proof The correctness of the algorithm is obvious.

Step 1 requires $O(log\ n)$ units of time and makes use of $O(n)$ processors, from Theorem 4.2.. Step 4 requires $\log k = O(log\ n)$ units of time and only one processor.

Loop 5-7 requires $O(log\ n)$ processors an $O(log\ n)$ units of time. Loop 2-9 requires $O(n^2/\log n)$ processors and $O(log\ n)$ units of time. □

5. Comparing Fingerprints

We now have n sets of finger prints F_i, $1 \leq i \leq n$ with $F_i = \{ f_{s,i} : s \in S \}$. Two elements from S are in the same set in the final partition if and only if they have the same finger print in every set F_i, $1 \leq i \leq n$. A formal recursive definition of these refinements , has as follows:

$$\pi_k = \{B_l^{(k)} := B_m^{(k-1)} \cap f^{-k}(A_j)\ ,\ \text{for some}\ \ m,j\ \} \tag{5.1}$$

Also in the sequel $B_m^{(k-1)}$ is said to be the *parent set* of $B_l^{(k)}$ if and only if $B_l^{(k)} \subseteq B_m^{(k-1)}$.

Consider the sets of fingerprints $F_1, F_2, \cdots, F_n$. Each element in S has a finger print corresponding to each F_i. We can construct a vector $(f_{s,1}, f_{s,2}, \cdots, f_{s,n})$ which is a list of each of these corresponding finger prints. We can sort these vectors lexicographicly, using the parallel algorithms given in [3].

5.1. Theorem [3]

There exists a parallel algorithm on CREW PRAM that lexicographicly sorts n words all of length l over an alphabet Σ of size $O(n)$ in

$$O\left(\frac{\log nl}{\log nl/p}\ \frac{nl}{p}\right)$$

units of time and uses $p \leq nl$ processors. □

Therefore given the sets F_1 , ... F_n, the computation of the final partition can be done in $O(log\ n)$ units of time and using $O(n^2)$ processors. One may observe that Theorem 5.1 yields an $O(n^2)$ cost algorithm, that requires $O(n^x)$ units of time and $O(n^{2-x})$ processors, for any $0 \leq x \leq n$, thus improving the cost.

6. The Overall Algorithm, its Analysis And Correctness

The pseudo-code below gives the overall algorithm that was outlined in section 2; it is a combination of algorithms given in section 3, 4 and 5.

6.1. Algorithm

begin

Compute $f^k(s)$, $1 \leq k \leq n$, for all s in S using algorithm 4.3;

for $m = 1$ **to** n **pardo**

Compute $f_{s,m}$ for all s in S, using algorithm 3.2;

Compute the final partition as in section 5, using algorithm 5.1;

odpar

end. □

In order to show the correctness of algorithm 6.1, we first prove the following three lemmas:

6.2. Lemma

Let $B_l^{(k)}$'s be as in (5.1). Then we have that:

(i) $f^{-k}(A_i) = \bigcup_{l \in L_i} B_l^{(k)}$ for some set of indices L_i, $1 \leq i \leq k$.

(ii) There is no pair $B_{l_1}^{(k)}$, $B_{l_2}^{(k)}$ in (i) with the same parent set.

Proof (i) Obvious by the definition of $B_l^{(k)}$'s in section 5.

(ii) Proof by induction. One can see that

$$f^{-1}(A_i) = \bigcup_{l \in L_i} B_l^{(1)}, \quad 1 \le i \le k$$

and that each $B_l^{(1)}$ has A_{j_l}, for some j_l, as parent set and all parent sets are distinct.

Now assume that (ii) holds for $k = m$. Then we have

$$f^{-m-1}(A_j) = f^{-1}(f^{-m}(A_j)) = \bigcup_{l \in L'} (f^{-1}(B_l^{(m)}))$$

Assert that $B_{l_1}^{(m+1)}$ and $B_{l_2}^{(m+1)}$ are children of $B_l^{(m)}$. Then we have that

$$B_{l_1}^{(m+1)} \subseteq f^{-1}(B_{l_1}^{(m)}) \subseteq f^{-1}(B_{l_1}^{(m-1)}) \tag{6.1}$$

$$B_{l_2}^{(m+1)} \subseteq f^{-1}(B_{l_2}^{(m)}) \subseteq f^{-1}(B_{l_2}^{(m-1)}) \tag{6.2}$$

where $B_{l_1}^{(m-1)}$ and $B_{l_2}^{(m-1)}$ are the parent sets of $B_{l_1}^{(m)}$ and $B_{l_2}^{(m)}$ respectively.

One can see that from (6.1) and (6.2) is implied that $B_l^{(m)}$ has been splitted from the refinement with respect to f^{-m}, a contradiction.□

6.3. Lemma

The following holds:

$$B_l^{(k)} = B_m^{(k-1)} \cap f^{-1}(B_r^{(k-1)}), \quad 1 \le k \le n$$

for some l, m, r.

Proof From the definition of $B_l^{(k)}$ in (5.1) we have

$$B_l^{(k)} = B_m^{(k-1)} \cap f^{-k}(A_i) = B_m^{(k-1)} \cap f^{-1}(\bigcup_{l \in L} B_l^{(k-1)})$$

Suppose that there exist a, b in $B_m^{(k-1)}$ such that $a \in f^{-1}(B_{l_1}^{(k-1)})$ and $b \in f^{-1}(B_{l_2}^{(k-1)})$ for some l_1, l_2 in L. Then $a \in f^{-1}(B_{l_1}^{(k-2)})$ and $b \in f^{-1}(B_{l_2}^{(k-2)})$, where $B_{l_1}^{(k-2)}$ and $B_{l_2}^{(k-2)}$ are the parent sets of $B_{l_1}^{(k-1)}$ and $B_{l_2}^{(k-1)}$ respectively. But refining with respect to f^{-1} at the $(k-1)$-th level it would have to locate a and b, into different sets - a contradiction. Therefore there is at most one $r \in L$ such that

$$B_m^{(k-1)} \cap f^{-1}(B_r^{(k-1)}) \neq \emptyset \quad .\square$$

6.4. Lemma

Let $\pi_0 = \{B_1^{(0)}, \cdots, B_k^{(0)}\}$ be a partition of S. Then let

$$B_i^{(k)} = B_{j_i}^{(k-1)} \cap f^{-1}(B_{l_i}^{(k-1)}) \neq \emptyset \quad \text{for some } l_i \tag{6.3}$$

Then

$$\pi_n = \{B_i^{(n)} : \text{for all } i\}$$

is the coarsest refinement of π that respects the function f.

Proof

One can see that $|\pi_{k+1}| > |\pi_k|$ or $|\pi_{k+1}| = |\pi_k|$. If the later occurs, then easily follows that $|\pi_k| = |\pi_{k+1}| = |\pi_{k+2}| = \cdots$. Also since the cardinality of S is n, this will occur for some $k \leq n$.

Assume that π_n does not respect the function f. Then we have that

$$f(B_i^{(n)}) \text{ 'o/} \subseteq B_j^{(n)} \text{ and } f(B_i^{(n)}) \cap B_j^{(n)} \neq \emptyset, \quad \text{for some } i, j$$

which implies that

$$B_i^{(n)} \subseteq f^{-1}(B_j^{(n)}) \text{ and } B_i^{(n)} \cap f^{-1}(B_j^{(n)}) \neq \emptyset$$

and thus $|\pi_{n+1}| > |\pi_n|$ a contradiction.□

6.5. Theorem

Algorithm 6.1 correctly computes the coarsest refinement of π with respect to the function f.

Proof It follows from lemmas 6.2, 6.3 and 6.4. □

6.6. Theorem

Algorithm 6.1 requires $O(log\ n)$ units of time and uses $O(n^2)$ processors.

Proof It follows from Theorems 3.3, 4.4 and 5.1.□

Following the remarks below Theorem 5.1 one can show :

6.7. Theorem

There exists an $O(n^2)$ cost algorithm on CREW PRAM for the one function coarsest partition problem.□

7. Conclusions

Paige and Tarjan [5] gave an $O(n)$ sequential algorithm for the single-function partition problem; their analysis of the problems may lead to better than $O(n^2)$ cost parallel algorithms. But the existence of an optimal cost $O(n)$ parallel algorithm is still an open problem together with the question whether the "many-functions coarsest partition problem" (see [1]) belongs in NC. Answers to these questions may help in understanding of the parallel behaviour of problems closely related to partitioning like doubly lexical ordering, chordality of a graph and relational partitioning (see [5]).

8. References

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

[2] 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.

[3] Iliopoulos, C.S., *Optimal cost parallel algorithms for lexicographical ordering*. Purdue University, Tech. Rep. 602, (1986)

[4] Paige, R., Tarjan, R., *A linear time algorithm to solve the single function coarsest partition problem*, IIth ICALP, Lecture notes in Computer Science 172, Springer, Berlin, 1984.

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

[6] Paige, R., Tarjan, R., *Three efficient algorithms Based on Partition refinement,* manuscript.

[7] Downey, P., Sethi, R., Tarjan, R., "Variations on the Common Subexpression Problem", JACM 27, 4 (1980) 758–771.