1986

Optimal Cost Parallel Algorithms for Lexicographical Ordering

Costas S. Iliopoulos

Report Number:
86-602

OPTIMAL COST PARALLEL ALGORITHMS FOR LEXICOGRAPHICAL ORDERING

Costas S. Iliopoulos

CSD-TR-602
May 1986

OPTIMAL COST PARALLEL ALGORITHMS

FOR

LEXICOGRAPHICAL ORDERING

by

Costas S. Iliopoulos

Purdue University

Dept. of Computer Science

West Lafayette, IN 47907

ABSTRACT

Optimal cost parallel algorithms for lexicographical ordering on a CREW PRAM are presented here. An $O(\frac{\log n}{\log (n/p)} \frac{n}{p})$ algorithm for sorting n integers from the range $\{1, \cdots, n\}$ using $p \leq n$ processors is given here. Also an algorithm for sorting n strings of size l over an alphabet of size s is presented, that requires $O(\frac{\log nl}{\log (nl/p)} \frac{nl}{p} + \frac{s}{p})$ units of time and it makes use of $p \leq \min \{nl/\log l, s/\log s\}$ processors. Both algorithms are of optimal cost with $p \leq n^x$ and $p \leq \min \{(n\ l)^{1-x}, s/\log s\}$ for any $0 < x < 1$ respectively.

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 processors. We shall focus on a subclass of problems in P, the set LINEARTIME - i.e. the set of problems solvable in time proportional to the size of the input. We are interested in characterizing the computational behaviour of problems in LINEARTIME on parallel models of computation.

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. A family of optimal parallel algorithms for a problem in LINEARTIME is defined to be the one that satisfies

$$p\,t = O(n) \quad \text{for} \quad 1 \le p \le p_o(n) \le n \tag{1.1}$$

with $p_o(n)$ close to n. Note that an algorithm using p_o processors satisfying suffices to define the family since if we leave only p processors available then each processor simulates p_o/p processors and still (1.1) holds. We denote by ZC (Zvi's (Galil) class see [6]) the subclass of problems in LINEARTIME that have optimal cost parallel algorithms for CREW PRAM. Currently only two non-trivial problems have been shown to be in ZC, the Selection problem (see [2] and [8]- a parallel version of [3]) and String Matching (see [6]).

Here we present an algorithm for sorting a set of n integers from the range $\{1, \cdots, s\}$ that requires $O(\frac{\log n}{\log n/p}\ \frac{n}{p} + \frac{s}{p})$ units of time and makes use of p processors with $p \le \min\{\, n\ ,\ s/\log s\ \}$. This algorithm has optimal cost $t\,p = O(n+s)$ for $1 \le p \le \min\{\, n^{1-x}\ ,\ s/\log s\ \}$, for some $0 < x < 1$. The only known optimal cost algorithm is a probabilistic algorithm due to Reif in [7]. Furthermore in the case that the range of the integers is $\{1, \cdots, n\}$, the above family of algorithms includes an algorithm that requires $O(\log n)$ units time by using n processors, matching the Ajtai-Kolmos-Szemeredi sorting network bounds in [1].

Also we present an algorithm for lexicographic ordering that requires $O(\frac{\log nl}{\log nl/p}\ \frac{nl}{p} + \frac{s}{p})$ and makes use of p processors with $p \le \min\{\, nl/\log l\ ,\ s\ /\log s\ \}$,

where n is the number of strings given, s is the size of the alphabet and l the size of the strings (all of equal size). The above algorithm is of optimal cost

$$t\ p = O(nl + s) \text{ for } 1 \le p \le \min\{(nl)^{1-x}, s/\log s\}, \text{ for some } 0 < x < 1$$

Furthermore under the constraint that the size of alphabet is bounded by n, we show an optimal cost algorithm with complexity

$$(t_0, p_0) = (n^x \log l, n^{1-x} l/\log l) \text{ for any } 0 < x < 1$$

All algorithms presented here can be adapted to the EREW PRAM (Exclusive Read-Exclusive Write PRAM - as CREW PRAM with the constraint that two processors can not access the same memory location concurrently) without significant change of their asymptotic complexity.

2. SORTING A SEQUENCE OF INTEGERS FROM A BOUNDED RANGE

Here we consider the lexicographical ordering problem in the restricted form that all strings are length 1. The restricted problem has as follows:

INPUT: A sequence of integers $\alpha_1, \cdots, \alpha_n$ from the set $\{1, \cdots, s\}$.

OUTPUT: A permutation σ on n points such that

$$\alpha_{\sigma(i)} \ge \alpha_{\sigma(i+1)} \quad \text{for} \quad 1 \le i < n$$

2.1. An Outline Of The Algorithm And The Main Procedures

We assume that all common memory positions are set to zero. The method has as follows:

(i) (Initialization) Here we make use of an $(\frac{n}{p}) \times s$ matrix M, where p is the number of processors available . Each processor P_r, for $1 \le r \le p$ is assigned a subset S_r of $O(n/p)$ elements of the input sequence $\alpha_1, \cdots, \alpha_n$ and for each element q in S_r , it augments the entry m_{rq} by 1. The output will consist of:

(1) the values m_{rq} for $1 \le r \le n$, $1 \le q \le s$ of the matrix M, where m_{rq} denotes the number of the occurrences of the element q in the subset S_r.

(2) a list INDEX (r) for each row, containing the indices of all the columns that an entry occurred.

(ii) Compacting the columns. We partition the matrix M into submatrices that have n / p rows of M - called "n/p strips" in the sequel - and we compact all n/p strips in parallel; then recursively apply the method to the reduced by a factor n/p output matrix. We compact an "n/p strip" into a single row $(m'_{i1}, \ldots, m'_{is})$ by doing for each row of the (n/p) strip the following:

(1) We partition the set INDEX(r) into subsets S_k of size $\lceil n / p \rceil$ and we assign a processor P_k to each subset S_k.

(2) In parallel each processor P_k updates the corresponding columns of M' using the m_{iq}'s for each $q \in S_k$.

(iii) Compacting the row. The result matrix M' above will consist of a single row that can be compacted by means of the "partial sum tree" method (see Willie [9] and [5]) in $O(s/p)$ units of time, where $p \leq s / \log s$ is the number of processors used.

The procedure for the initialization step (i) above has as follows:

begin

$l \leftarrow \lceil n / p \rceil$;

forall $1 \leq r \leq p$ **pardo**

for $j = rl$ **to** $(r+1)l$ **do**

if $m_{r\alpha_j} = 0$ **then** INDEX $(r) \leftarrow$ INDEX $(r) \bigcup \{\alpha_j\}$;

$m_{r\alpha_j} \leftarrow m_{r\alpha_j} + 1$;

od

comment The list INDEX yields the set of all distinct integers that appear in $\{\alpha_{rl}, \cdots, \alpha_{(r+1)l}\}$

odpar

end. □

The procedure for compacting the columns step (ii) has as follows:

Procedure COMPCOL (M)

Initialize as it was described above;

forall "n / p-strips" **pardo**

Let the current srip be the q-th one ;

for every row r of the q-th strip **do**

forall $t \in$ INDEX(r) **pardo**

$m'_{qt} \leftarrow m'_{qt} + m_{rt}$ for every row in the q-th strip;

INDEX $(q) \leftarrow$ INDEX $(q) \bigcup \{t\}$;

odpar

od

COMPCOL(M') ;

odpar. □

2.2. Analysis Of The Algorithm

The complexity and the number of processors required from the above method is given by the following theorem :

THEOREM 2.1 There exists a family of parallel algorithms that sorts a sequence of n integers taken from the set $\{1, \cdots, s\}$ in $O(\frac{\log n}{\log (n/p)} \frac{n}{p} + \frac{s}{p})$ units of time and using $p \leq \min \{ n , s / \log s \}$ processors.

Proof

We shall analyze the algorithm in three stages: (i) initialization, (ii) compacting the columns and (iii) compacting the rows.

(i) In the initialization one processor is assigned to a set of elements of cardinality $\lceil n / p \rceil$, subset of the input set and for each α_j of this subset, it increases the α_j entry of the associated row of the matrix M. One can see that this requires $O(n / p)$ units of time and p processors.

(ii) One may observe that the list INDEX (r) is always in a compact format and if INDEX$^{(k)}(r)$ denotes INDEX (r) at the k-th recursive call, then its size is dominated by

$$|\text{INDEX}^{(k)}(r)| \leq \lceil (n/p)^k \rceil$$

Therefore the number of processors needed for each strip equals to $\lceil (n/p)^k \rceil$ at the k-th recursive call. Moreover one can see that the number "n / p-strips" at the k-th recursive call equal to $n / (n/p)^{k+1}$, and thus the number of processors needed at a recursive call of the procedure equals to p.

One can see that one recursive call requires $O(n / p)$ units of time and that the number of recursive calls is bounded by $\frac{\log n}{\log (n/p)}$. Therefore COMPCOL requires $O(\frac{\log n}{\log (n/p)} \frac{n}{p})$ units of time in the worst-case.

(ii) From [9] one can see that the compacting of a list by using $p \leq s / \log s$ processors can be done in $O(s/p)$ units of time.

From the above analysis the theorem follows. □

COROLLARY 2.2 There exists a parallel algorithm for sorting n integers from the range 1 to n in $O(\frac{\log n}{\log (n/p)} \frac{n}{p})$ units of time using $p \leq n$ processors.□

COROLLARY 2.3 There exists an optimal cost algorithm for sorting n integers in the the range $\{1,....,n\}$ in $O(n^x)$ units of time using n^{1-x} processors, for any $0 < x < 1$.□

3. LEXICOGRAPHICAL ORDERING

Here we consider several instances of the following problem:

INPUT: A set of strings

$$x_i = (a_{1i}, a_{2i}, \cdots, a_{li}), \quad 1 \leq i \leq n$$

of length l over a totally ordered alphabet Σ, where the cardinality $|\Sigma|$ of Σ equals to s.

OUTPUT: A sequence of strings $x_{\sigma(i)}$, $1 \leq i \leq n$, such that

$$x_{\sigma(i)} \leq x_{\sigma(i+1)} \tag{3.1}$$

The relation (3.1) holds, if and only if, there exists an integer q such that $\alpha_{q\sigma(i)} < \alpha_{q\sigma(i+1)}$ and for all $j \leq q$, $\alpha_{j\sigma(i)} = \alpha_{j\sigma(i+1)}$.

Below, we present an algorithm for lexicographical sorting in the case of the size s of the alphabet Σ is an arbitrary variable, and an algorithm for the case that s is a polynomial in terms of n.

3.1. The Algorithm For Arbitrary Size Alphabets

As first step we compute the subalphabets A_m, for $1 \leq m \leq l$, i.e., the ordered set of symbols that occur in the m-th position of the strings $x_1, \cdots, x_n$. The computation of the A_m's, will provide us with a bounded alphabet to work with, avoiding computations involving the alphabet Σ whose cardinality s might be very large.

begin

Sort $\{(k, \alpha_{ki})$ for $1 \le k \le l$ and $1 \le i \le n\}$; (3.2)

comment Preserving the order of the pairs (k,α_{ki}) by the first component, we sort the pairs by the second component using the algorithm of Theorem 2.1.

$A_k \leftarrow \{(k, a_{ki})\}$ for $1 \le k \le l$;

comment The computation of the subalphabet A_m is done by removing the duplicates of the sorted list $\{(k,\alpha_{ki})\}$ by a generalization of the "partial sum tree" algorithm (see [9]).

end . □

Using Theorem 2.1, one can show that :

LEMMA 3.1 The computation of the subalphabets above requires $O(\frac{\log nl}{\log (nl/p)} \frac{nl}{p} + \frac{s}{p})$ units of time, where p is the number of processors used with $p \le \min \{nl, s/\log s\}$ □

As initialization procedure we partition the strings x_i, for $1 \le i \le n$ into substrings w_{ji} , $1 \le i \le n$, $1 \le j \le p' := \lceil \sqrt{p} \rceil$, such that

$$x_i = w_{1i}\, w_{2i} \cdots w_{p'i} \quad \text{with } |w_{ji}| = \lceil l/p' \rceil =: q$$

and then we sort the lists $L_j = \{w_{ji}: 1 \le i \le n\}$ for $1 \le j \le p'$ in parallel. The choice of $p' = \sqrt{p}$ follows the facts that the number of lists L_j's is $\sqrt{p}$ and the sorting of each list requires $\sqrt{p}$ processors, thus the total number of processors needed is p equal to the number of processors available. A procedure for sorting one of these lists is given below:

procedure INITIALSORT [$w_i = (\alpha_{1i}, \cdots, \alpha_{qi})$, $1 \le i \le n$]

$w_i \leftarrow \alpha_{1i}$, for $1 \le i \le n$;

for $k = 2$ **to** q **do**

Sort $(w_1, \alpha_{k1}), \cdots, (w_n, \alpha_{kn})$ over the alphabet A_k as in (3.2) ;

$w_i \leftarrow w_i\, \alpha_{ki}$ for $1 \le i \le n$;

od .□

It is not difficult to show that :

LEMMA 3.2 The procedure INITIALSORT requires $O(\frac{\log n}{\log(n/\sqrt{p})} \frac{nl}{p})$ units of time, where $\sqrt{p} \leq n$ is the number of processors used. □

We shall make use of a recursive procedure called LEXSORT that having as input a set of strings, return the set sorted. An outline of the procedure has as follows :

(i) If the strings to be sorted are of length $\lceil l/\sqrt{p} \rceil$ we use INITIALSORT to sort them, otherwise

(ii) We partition each string of the list into two substrings of equal length, thus creating two lists of strings that we sort in parallel.

(iii) If the strings in the initial list were $x_i = y_i\ w_i$, $1 \leq i \leq n$ then we sort them by sorting the pairs (y_i, w_i) over the alphabet $\bigcup_{i=1}^{n} \{w_i\}$.

The pseudo-code below describes the method in detail.

Procedure LEXSORT $[(a_{1i}, \cdots, a_{mi})$ for $1 \le i \le n$, for some $1 \le m \le l]$

begin

if $m = \lceil \log l \rceil$ **then return** INITIALSORT $[(a_{1i}, \cdots, a_{mi}), 1 \le i \le n]$;

else $k \leftarrow \lceil m/2 \rceil$;

$\{y_1, \cdots, y_n\} \leftarrow$ LEXSORT $[(a_{1i}, \cdots, a_{ki}), 1 \le i \le n]$; (3.3)

$\{w_1, \cdots, w_n\} \leftarrow$ LEXSORT $[(a_{k+1,i}, \cdots, a_{mi}), 1 \le i \le n]$; (3.4)

comment The above two steps are executed in parallel and the y_i's and w_j's represent sorted sequences of strings of length k.

Let w_j' for $1 \le j \le n$: $x_{i_j} = q_j y_j w_j' p_j$ such that $|w_j'| = m - k$,for some strings q_j, p_j.

$A \leftarrow$ the sorted sequence of w_i, $1 \le i \le n$, without duplicates; (3.5)

Sort $\{(y_j, w_j'): 1 \le j \le n\}$ over the alphabet $\{w_1, \cdots, w_n\}$; (3.6)

return the sorted sequence $y_{\sigma(j)}\, w'_{\sigma(j)}$.

end. □

THEOREM 3.2 There exists a family of parallel algorithms for lexicographical ordering that requires $O(\frac{\log nl}{\log nl/p}\ \frac{nl}{p} + \frac{s}{p})$ units of time and makes use of p processors with $p \le \min\ \{\ nl/\log l\ ,\ s\ /\ \log s\ \}$.

Proof

Step (3.5) requires $O(\ n/\sqrt{p}\ )$ units of time and makes use of $\sqrt{p} \le n\ /\ \log n$ processors for using "partial sum tree" computation.

Step (3.6) requires $O(\frac{\log n}{\log(\ n/\sqrt{p}\ )}\ \frac{n}{\sqrt{p}})$ units of time and $\sqrt{p} \le n\ /\log n$ processors by using the algorithm of Corollary 2.2, and the fact that $|A\ | \le n$.

The number of recursive calls is at most $O(\log l)$. Therefore the time required is bounded by

$$O\left(\frac{\log n}{\log n/\sqrt{p}}\ \frac{nl}{p} + \frac{\log n}{\log n/p}\ \frac{n}{p}\log l\ \right)$$

using Lemma 3.2 for the time analysis of INITIALSORT. Furthermore using Lemma 3.1 one can show the overall time required for lexicographical sorting.

At the parallel execution of steps (3.3) and (3.4) one can see that we can have at most $\sqrt{p}$ LEXSORT calls running in parallel and each requires $\sqrt{p}$ processors (for INITIALSORT). Therefore the number of processors required is at most p . □

COROLLARY 3.3 There exists a family of optimal algorithms for lexicographic ordering, whose complexity satisfies the equation

$$t\ p = O(nl + s),\ \text{for} p \le \min\{(n\ l)^{1-x}, s\ /\log s\ \}\text{for some}\, 0 < x < 1$$

where t and p denote the time and the number of processors respectively. □

3.2. The Algorithm For Bounded Size Alphabets

Here we assume that the size of the alphabet Σ is $s = O(n)$. By modifying the computation of the subalphabets A_m for $1 \le m \le n$, we can improve the complexity of the lexicographical problem. The computation of the subalphabets can be done as follows.

begin

forall $j = 1$ **to** $\sqrt{p}$ **pardo**

for $k = jl/\sqrt{p}$ **to** $(j+1)l/\sqrt{p}$ **do**

Sort $\{(k, \alpha_{ki}), 1 \le i \le n\}$ as in (3.2);

$A_k \leftarrow \{a_{ki}, 1 \le i \le n\}$ as in (3.2);

od

odpar

end

One can show that the above computation requires $O\left(\frac{\log n}{\log n/\sqrt{p}}\ \frac{n\ l}{p}\right)$ units of time and makes use of $\sqrt{p}$ processors.

By modifying the lexicographical algorithm of Theorem 3.1 one can show:

THEOREM 3.4 There exists a parallel algorithm for the lexicographical ordering over an alphabet of size $O(n)$ that requires $O(\frac{\log n\ l}{\log n\ l/p}\ \frac{nl}{p})$ with $1 \leq p \leq nl/\log l$. □

COROLLARY 3.5 The above algorithm has optimal cost for $p \leq n^{1-x}l/\log l$, for any $0 < x < 1$, requiring $O(n^x \log l)$ units of time.□

4. CONCLUSIONS

The problems that we encounter in designing an algorithm on a CREW PRAM for lexicographical ordering can be summarized as follows:

(i) Sorting n integers over the range $\{1, \cdots, n\}$. We shall surprised to see a subpolynomial (in terms of time) optimal algorithm for this problem. It is not difficult to see that any improvements on the parallel complexity of the integer sorting problem will reflect directly to the complexity of the lexicographical ordering problem. Also it worth mentioning that for p=n our algorithm of Corollary 2.2 matches the cost of the sorting network of Aljai-Kolmos-Szemerendi in [1].

(ii) The arbitrary size s of the alphabet Σ makes necessary the computation of the subalphabets A_l (see section 3.1) that have size $O(n)$; this allows us to make use the bounds of Corollaries 2.2 and 2.3 instead of Theorem 2.1 that would lead to a non-optimal cost of $O(sl + nl)$.

(iii) The existance of an optimal cost algorithm for the problem of sorting a set of strings of variable length is an open question.

5. REFERENCES

[1] Ajtai,M.,Kolmos, J., and Szemeredi,E., *An O(n log n) sorting network* Combinatorica 3, 1 (1983), pp 1-19

[2] Akl, S., *An optimal algorithm for Selection* Info.Proc.Letters 19(1984)47-50

[3] Blum, M., Floyd, R.W., Pratt, V.R., Rivest, R.L., and Tarjan, R.E., *Time bounds for selection* J. Computer and System Sciences , 7, 4 (1972) , pp 448-461.

[4] Cole,R., Yap, C.K., *A parallel median algorithm* Info.Proc.Letters 20 (1985) 137-139

[5] Dekel,E., Sahni,S., *Binary trees and parallel scheduling algorithms* IEEE Transactions on Computers, Vol 32, 3, (1983)

[6] Galil, Zvi, *Optimal parallel algorithms* in VLSI: Algorithms and Architectures, Bertolazzi and Luccio F. (Editors) North-Holland (1985)

[7] Reif, J.H., *Optimal parallel algorithms for integer sorting and graph connectivity* 26-th Symposium on Foundations of Computer Science, (1985)

[8] Viskin, U., *An optimal parallel algorithm for selection* Manuscript, Courant Inst., (1983)

[9] Wyllie, J.C., *The complexity of parallel computations* Ph.D. Thesis, Cornell Univ., N.Y., (1979)