COMPUTER SCIENCE PUBLICATION

EFFICIENT PARALLEL ALGORITHMS FOR SOME SUBSEQUENCE PROBLEMS

F.Y.L. Chin, W.W. Tsang and T.W. Lam

Technical Report TR-89-03

March 1989

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF HONG KONG
POKFULAM ROAD
HONG KONG

Efficient Parallel Algorithms for Some Subsequence Problems

Francis Y.L. Chin, Wai Wan Tsang and Tak W. Lam*

Department of Computer Science
University of Hong Kong
Pokfulam Road, Hong Kong

Abstract

Both subsequence problems, finding longest common subsequence of two strings of length m and n, and one-dimemsional pattern matching of n input elements, have sequential dynamic programming algorithms which take $O(mn)$ and $O(n)$ time, respectively. In this paper, we show that the former problem takes $O(\log m \log n)$ time using $nm^e/(\log m \log n)$ processors, where e is the exponent of matrix multiplication in semiring, while the latter problem, as well as some similar ones, can be solved optimally in $O(\log n)$ time using $n/\log n$ processors, both on parallel random access machines (PRAM).

Keywords: Parallel Algorithm, PRAM, Dynamic Programming, Subsequence Problems.

1 Introduction

Dynamic Programming (DP) [Bel,Tho] is one of several problem-solving techniques which are widely used in Computer Science and Operations Research. As parallel computation [KaRa,Coo,Wyl,TsLaCh] becomes increasingly important in Computer Science, much effort has been devoted in designing parallel algorithms for those problems which can be solved with efficient sequential DP algorithms [LiWa,EdWa,LiLo,Ryt].

One main feature underlying DP is that the computation is performed in stages where operations of a stage make use of the results of previous stages. An optimal solution is found when the last stage is reached. Since this process is sequential in

*e-mail address: hkucs!chin@uunet.uu.net, hkucs!tsang@uunet.uu.net, hkucs!twlam@uunet.uu.net nature, a problem solved using DP may have to be expressed in a different form for efficient parallel computation. In this paper, we illustrate how to design polylog-time algorithms on parallel random access machines (PRAM) [FoWy] for solving some subsequence problems with sequential DP algorithms, such as finding longest common subsequence (LCS)[Hir75], one-dimensional pattern matching (1DPM)[Ben], finding longest nondecreasing (nonincreasing) subsequence (LNS) and maximum positive subsequence (MPS).

The LCS problem, which takes two input strings of length m and n, can be solved sequentially in $O(mn)$ time using DP technique [Hir75]. In Section 2, we describe an $O(\log m \log n)$ time algorithm using $nm^{\epsilon}/(\log m \log n)$ processors on a concurrent-read, exclusive-write parallel random-access machine (CREW PRAM), where ϵ is the exponent of matrix multiplication in semiring, i.e., the product of two $n \times n$ matrices can be computed using no more than $O(n^{\epsilon})$ operations. In Section 3, we show that the 1DPM problem is a special case of the sum-range-product problem. These two problems which have to consider $n(n-1)/2$ ranges of n elements, i.e., $x_j * \ldots * x_k$ where $1 \leq j \leq k \leq n$, can be solved by the DP technique in $O(n)$ time sequentially. In this paper, we devise an algorithm to solve the sum-range-product problem optimally in $O(\log n)$ time using $n/\log n$ processors on an exclusive-read, exclusive-write parallel random-access machine (EREW PRAM). The LNS and MPS problems can also be solved in the similar way. Some remarks on designing parallel algorithms for problems with sequential DP algorithms are given in Section 4.

2 The Longest Common Subsequence Problem

Given a string X, a *subsequence* of X is a string which can be obtained by deleting characters from X, e.g., "seen" is a subsequence of "sequence". The *longest common subsequence* problem can be stated as follows: Given strings $X = x_1 \cdots x_n$ and $Y = y_1 \cdots y_m$ over a finite alphabet, find a string $Z = z_1 \cdots z_p$ such that Z is a common subsequence of X and Y with maximum length. A DP algorithm has been designed to solve the LCS problem sequentially in $O(mn)$ time [Hir75]. [AhHiUl] has proved a lower bound of $\Omega(mn)$ for the LCS problem over an alphabet of three or more symbols if only equal-unequal comparisons are used. For some particular input strings, algorithms that take less than $O(mn)$ time have been designed [HuSz,Hir77]. This problem has attracted much research attention and has a number of applications [WoCh,Ita,FiMa].

Parallel algorithms for the LCS problem have been studied in [ChRo, EdWa]. In [EdWa], a parallel algorithm that runs in $O(n+m)$ time using $m+1$ processors on a systolic-array-like machine is presented. In [ChRo], a constant time algorithm has been developed on a model called bus automaton (BA), which can be viewed as a cellular automaton with a locally switchable global communication network. Instead

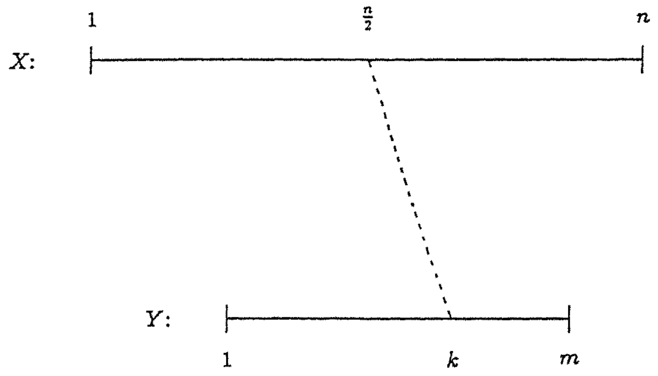

Figure 1: Decomposition of the LCS Problem

of using a powerful parallel computation model such as BA, the following subsections describe an $O(\log m \log n)$ algorithm which runs on a CREW PRAM.

2.1 Main idea

Our method uses a dynamic programming formulation different from [Hir75] to solve the LCS problem in parallel. As shown in figure 1, an LCS of X and Y can be formed by concatenating an LCS of $x_1 \cdots x_{\frac{n}{2}}$ and $y_1 \cdots y_k$ and that of $x_{\frac{n}{2}+1} \cdots x_n$ and $y_{k+1} \cdots y_m$, for some value k, where $0 \leq k \leq m$, such that the resulting LCS is the longest possible. The subproblems so decomposed can be broken down recursively in the same way until they are small enough to be solved directly.

Let $L(r,s,i,j)$ be the length of an LCS of $x_r \cdots x_s$ and $y_i \cdots y_j$. $L(1,n,1,m)$ is then the length of an LCS of X and Y. The following is the recurrence relation for the L's of the subproblems derived from the decompositon. Assume $1 \leq r < s \leq n$, $1 \leq i \leq j \leq m$,

$$L(r,r,i,j) = \begin{cases} 1 & \text{if } x_r \text{ matches with one of } y_i, \cdots, y_j \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$L(r,s,i,j) = \max_{i-1 \leq k \leq j} \{L(r, \lfloor \frac{r+s}{2} \rfloor, i, k) + L(\lfloor \frac{r+s}{2} \rfloor + 1, s, k+1, j)\} \tag{2}$$

The above recurrence assumes that n is a power of 2 and $L(r,s,i,j) = 0$ if $i > j$.

2.2 Computation of the L values

The following describes how $L(1,n,1,m)$ is computed in parallel. Contrast to the top-down decomposition of the LCS problems suggested by the recurrence, the computation is performed stage by stage in a bottom-up manner.

A set-up is needed before the beginning of the first stage. For all $1 \leq r \leq n$, $1 \leq j \leq m$, $L(r,r,j,j)$ is set to 1 if x_r matches y_j, otherwise 0. This step can be completed in $O(1)$ time with mn processors. The next step is to compute $Q(r,j) = \sum_{1 \leq k \leq j} L(r,r,k,k)$ which gives the number of occurrences of x_r in $y_1 \cdots y_j$, and $Q(r,j) - Q(r,i-1)$ is the number of occurrences of x_r in $y_i \cdots y_j$. Since x_r can match with at most one character in $y_i \cdots y_j$, the length of an LCS of x_r and $y_i \ldots y_j$ is at most 1. With the assumption that $Q(r,0) = 0$, we have $L(r,r,i,j) = \min\{1, Q(r,j) - Q(r,i-1)\}$ for $1 \leq r \leq n$ and $1 \leq i \leq j \leq m$.

For a fixed r, the computation of $Q(r,j)$ for $1 \leq j \leq m$ is a prefix sum problem which takes $O(\log m)$ time using $m/\log m$ processors [LaFi]. Thus, for $1 \leq r \leq n$, $1 \leq j \leq m$, $Q(r,j)$ can be computed in $O(\log m)$ time using $nm/\log m$ processors. Afterward, all the $L(r,r,i,j)$'s are computed in $O(1)$ time using nm^2 processors. Therefore, this set-up procedure can be completed in $O(\log m)$ time.

All the other L's are computed according to Eq(2) stage by stage. In particular, $L(2r-1, 2r, i, j)$'s, for $1 \leq r \leq \frac{n}{2}$, are computed in the first stage; $L(4r-3, 4r, i, j)$'s, for $1 \leq r \leq \frac{n}{4}$, in the second, etc. In general, $L(2^p(r-1)+1, 2^p r, i, j)$'s are computed in the pth stage for $1 \leq r \leq n/2^p$. Finally, $L(1,n,i,j)$'s, in particular $L(1,n,1,m)$, are computed in the $(\log n)$th stage.

To simplify the notion, let $u = 2^p(r-1)+1$, $v = 2^p r$ and $w = v - 2^{p-1}$. For $1 \leq i \leq j \leq m$, $L(u,v,i,j)$'s are computed from $L(u,w,i,j)$'s and $L(w+1,v,i,j)$'s, which are results of the $(p-1)$st stage. From Eq(2),

$$L(u,v,i,j) = \max_{i-1 \leq k \leq j} \{L(u,w,i,k) + L(w+1,v,k+1,j)\} \tag{3}$$

For fixed values of u and v, the computation of $L(u,v,i,j)$ for $1 \leq i \leq j \leq m$ is similar to matrix multiplication and can be performed in $O(\log m)$ time using $m^e/\log m$ processors, where e is an exponent of matrix multiplication in a semiring. As there are $n/2^p$ u-v pairs and the corresponding $L(u,v,i,j)$'s are computed in parallel in the pth stage, the total number of processors required is $nm^e/(2^p \log m)$. Since there are $\log n$ stages, the time complexity of the entire process is $O(\log m \log n)$.

According to the above analysis, the first stage requires $nm^e/2\log m$ processors. From the second stage onward, the number of processors required in each stage is half of that in the previous stage. This requirement is similar to a number of situations [ChLaCh,LaFi] and the same trick for processor scheduling can be applied to reduced the number of processors without a drawback in asymptotic time complexity. Thus, the L's can be computed in $O(\log m \log n)$ time using $nm^e/(\log m \log n)$ processors.

2.3 Backward tracing

A 3-D array, $\mathrm{D}[1\cdots 2n, 1\cdots m, 1\cdots m]$, is needed to keep track of the k values of Eq(2) so as to produce the LCS after $L(1,n,1,m)$ is found. In order to store up the additional information during the intermediate stages of computation, the first n values of the first index of D are reserved for information of the set-up procedure, the second $\frac{n}{2}$ values are for the first stage, the next $\frac{n}{4}$ values for the second stage, etc. The second and third indices in D correspond to the third and fourth indices of L.

At the completion of the set-up, $\mathrm{D}[r,i,j]$ stores the index k, such that y_k matches x_r for $i \le k \le j$ if $L(r,r,i,j) = 1$. At the completion of the pth stage, $\mathrm{D}[2n - n/2^p + r, i, j]$ stores the index k of Eq(3) which maximizes the value of $L(2^p(r-1)+1, 2^p r, i, j)$.

Upon the completion of computing $L(1,n,1,m)$, a backward process is started to find an LCS of X and Y. Let $k = \mathrm{D}[2n,1,m]$, we know that an LCS of X and Y can be formed by concatenating an LCS of $x_1\cdots x_{\frac{n}{2}}$ and $y_1\cdots y_k$ and an LCS of $x_{\frac{n}{2}+1}\cdots x_n$ and $y_{k+1}\cdots y_m$. By applying this strategy recursively, we can track down each component of the LCS of X and Y. With proper scheduling of processors, the backward process can be completed in $O(\log n)$ time.

Theorem 1 The LCS problem can be solved in $O(\log m \log n)$ time using $nm^e/(\log m \log n)$ processors on a CREW PRAM, where e is the exponent of the matrix multiplication in semiring.

3 The 1-D Pattern Matching and Sum-Range-Product Problem

Given n real numbers, $x_1, x_2, \ldots, x_n$, the *1-D pattern matching* problem [Ben] is to find the maximum sum in any contiguous subsequence of the input, i.e.,

$$\max_{1\le j\le k\le n} x_j + \ldots + x_k.$$

Let us consider the sample sequence given in [Ben], (31, -41, 59, 26, -53, 58, 97, -93, -23, 84), the answer of the 1DPM is 187 which is the sum of 59,...,97. This 1-D pattern matching problem is indeed a special case of the 2-dimensional one which provides the maximum likelihood estimator of a certain pattern in a digitized picture.

Let $*$ and $+$ be two associative and commutative operations on a domain D which follows distributive laws, i.e., $a*(b+c) = a*b + a*c$ and $(b+c)*a = b*a + c*a$. Given n elements, $y_1, \cdots y_n$, in D, the *sum-range-product* problem is to evaluate

$$A(n) = \sum_{1\le k\le n} \sum_{1\le j\le k} \prod_{j\le i\le k} y_i$$

Let D be the set of real numbers, $y_i = x_i$ for $1 \le i \le n$, $+$ be the maximum operation and $*$ be the real number addition. The 1DPM problem is then a special case of the sum range-product problem.

3.1 An optimal solution for the sum-range-product problem

In [Ben], Bentley described several approaches, such as preprocessing, divide-and-conquer and dynamic programming, for solving the 1-D pattern matching problem. In fact, there are $O(n^2)$ ranges of input numbers needed to be considered and a brute force method requires $O(n^3)$ time. Bentley finally gave a DP algorithm which solves the problem optimally in $O(n)$ time. Using a similar approach, we define $A(m)$ and $B(m)$ with the following recurrence,

$$B(m) = \begin{cases} 0 & \text{if } m = 0 \\ B(m-1) * y_m + y_m & \text{if } m > 0 \end{cases}$$

$$A(m) = \begin{cases} 0 & \text{if } m = 0 \\ A(m-1) + B(m) & \text{if } m > 0 \end{cases}$$

Note that, for $m = 1, \cdots, n$

$$B(m) = \sum_{1 \le j \le m} \prod_{j \le i \le m} y_i$$

$$A(m) = \sum_{1 \le k \le m} \sum_{1 \le j \le k} \prod_{j \le i \le k} y_i$$

The following theorem shows how to solve the sum-range-product problem in $O(\log n)$ time with $n/\log n$ processors on an EREW PRAM using the method developed by [GrLaPaGa] for linear recurrence computation.

Theorem 2 Assume that every $+$, $*$ operation takes unit time. The sum-range-product problem can be solved in $O(\log n)$ time using $n/\log n$ processors on an EREW PRAM.

Proof: The recurrence of $B(m)$ can be expressed in terms of matrix product $\beta_m = \beta_{m-1} * \psi_m$ where $\beta_m = [B(m)\ \ 1]$ and

$$\psi_m = \begin{bmatrix} y_m & 0 \\ y_m & 1 \end{bmatrix}$$

Thus $\beta_m = \beta_0 * \psi_1 * \ldots * \psi_m$, where $\beta_0 = [0\ \ 1]$.

Since $A(m)$ can also be expressed as $B(0) + B(1) + \ldots + B(m)$, the computation of $B(m)$ and $A(m)$ for $m = 1, \cdots, n$ is then reduced to two prefix-sum-like problems which can be solved in $O(\log n)$ time using $n/\log n$ processors on an EREW PRAM. Note that $A(n)$ is the solution of the sum-range-product problem. □

Corollary 3 The 1DPM problem can be solved in $O(\log n)$ time using $n/\log n$ processors on an EREW PRAM.

Proof: The 1DPM problem is a special case of the sum-range-product problem. □

Corollary 4 The range of the contiguous subsequence that produces the solution for the 1DPM problems can be obtained in $O(\log n)$ time using $n/\log n$ processors on an EREW PRAM.

Proof: Let $x_j, \cdots, x_k$ be the solution. Since $A(k) = A(k+1) = \ldots = A(n)$, k can be found easily given $A(m)$, for $m = 1, \cdots, n$, in a way similar to computing prefix sum. When k is known, j can be determined by finding the prefix sum of $x_k, x_{k-1}, \cdots, x_1$ such that the prefix sum corresponding to x_j is equal to $A(n)$. □

3.2 Some other subsequence problems

With the result for the sum-range-product problem, we show that optimal parallel algorithms also exist for some similar subsequence problems. For example,

1. *Longest Nondecreasing (Nonincreasing) Subsequence (LNS):*
   To find the maximum length of any nondecreasing (nonincreasing) contiguous subsequence.
   $$\max_{1 \le j \le k \le n} \{k - j + 1 \mid x_j \le \cdot \cdot \le x_k\}$$

2. *Maximum Positive Subsequence (MPS):*
   To find the maximum sum (product) of any contiguous subsequence of positive numbers.
   $$\max_{1 \le j \le k \le n} \{x_j + \ldots + x_k \mid x_j, \cdots, x_k > 0\}$$

Theorem 5 The LNS and MPS problems can be solved in $O(\log n)$ time using $n/\log n$ processors on an EREW PRAM.

Proof: These two problems is reducible to the sum-range-product problem. The reductions involved can also be done in $O(\log n)$ time using $n/\log n$ processors on an EREW PRAM.

1. LNS:
   Let D be the set of integers, $+$ be the maximum operation, $*$ be the integer addition. $y_1 = 1$ and, for $1 < i \le n$, $y_i = 1$ if $x_{i-1} \le x_i$, otherwise $-\infty$.

2. MPS:
   Let D be the set of real numbers, $+$ be the maximum operation, $*$ be the addition, for $1 \leq i \leq n$, $y_i = x_i$ if $x_i > 0$, otherwise $-\infty$.

   (To find the maximum product of any contiguous subsequence, set $y_i = 0$ if $x_i \leq 0$.)

□

4 Conclusions

Dynamic Programming is a powerful tool for solving problems efficiently. In this paper, we have successfully shown by examples that problems with DP formulations may be in NC [Coo] even though DP is implicitly sequential in nature. Two different approaches are used in designing parallel algorithms for this type of problems. The first one expresses the problem in a new DP formulation which can be computed in $O(\log n)$ stages. The second models the problem with a set of recurrence which can be evaluated efficiently in parallel due to the associativity of the operations involved. We intend to study more problems having sequential DP algorithms. The ultimate goal is to devise general principles for designing efficient parallel algorithms for this type of problems.

References

[AhHiUl] A.V. Aho, D.S. Hirschberg and J.D. Ullman, Bounds on the Complexity of the Longest Common Subsequence Problem, *J. Assoc Comp. Mach.,* Vol. 23, No. 1, January 1976, pp. 1-12.

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

[Bel] R. Bellman, *Dynamic Programming,* Princeton University Press, Princeton, New Jersey, 1957.

[Ben] J. Bentley, Programming Pearls - Algorithm Design Techniques, *Communications of the ACM*, Vol. 27, No. 9, September 1984, pp. 865-871.

[Bre] R.P. Brent, The Parallel Evaluation of General Arithmetic Expressions, *JACM*, Vol. 21, 1974, pp 201-208.

[ChLaCh] F.Y. Chin, J. Lam and I. Chen, Efficient Parallel Algorithms for Some Graph Problems, *CACM*, Vol. 25, No. 9, September 1982, pp. 659-665.

[ChRo] D.M. Champion and J. Rothstein, Immediate Parallel Solution of the Longest Common Subsequence Problem, *Proceedings of the 1987 International Conference on Parallel Processing,* 1987, pp. 70-77.

[Coo] S.A. Cook, A Taxonomy of Problems with Fast Parallel Algorithms, *Inform. and Control,* Vol. 64, 1985, pp. 2-22.

[EdWa] E. Edmiston and R.A. Wagner, Parallelization of the Dynamic Programming Algorithm for Comparison of Sequences, *Proceedings of the 1987 International Conference on Parallel Processing,* 1987, pp. 78-80.

[Fic] F.E. Fich, New Bounds for Prefix Circuits, *Proc. 15th Annual ACM STOC,* 1983, pp. 27-36.

[FiMa] W.M. Fitch and E. Margoliash, Construction of Phylogenetic Trees, *Science,* Vol. 155, 1967, pp. 279-284.

[FoWy] S. Fortune and J. Wyllie, Parallelism in Random Access Machines, *Proc. 10th Annual ACM STOC,* 1978, pp. 114-118.

[Gol] L.M. Goldschlager, A Unified Approach to Models of Synchronous Parallel Machines, *Proc. 10th Annual ACM STOC,* 1978, pp. 89-94.

[GrLaPaGa] A.C. Greenberg, R.E. Ladner, M.S. Paterson and Z. Galil, Efficient Parallel Algorithms for Linear Recurrence Computation, *Information Processing Letters,* Vol. 15, No. 1, August 1982, pp. 31-35.

[Hir75] D.S. Hirschberg, A Linear Space Algorithm for Computing Maximal Common Subsequences, *CACM,* Vol. 18, No. 6, 1975, pp. 341-343.

[Hir77] D.S. Hirschberg, Complexity of Common Subsequence Problems, *Lecture Notes in Comnputer Science, No 56, Fundamentals of Computation Theory,* Springer-Verlag, New York, 1977.

[HuSz] J.W. Hunt and T.G. Szymanski, A Fast Algorithm for Computing Longest Common Subsequences, *Comm. ACM,* Vol. 20, 1977, pp. 350-353.

[Ita] F. Itakura, Minimum Prediction Residue Principle Applied to Speech Recognition, *IEEE Trans. Acoust. Speech Signal Process.,* ASSP-23, 1975, pp. 67-72.

[KaRa] R.M. Karp and V. Ramachandran, A Survey of Parallel Algorithms for Shared-memory Machines, Report no. UCB/CSD 88/408, Computer Science Division (EECS), University of California Berkeley, California, March 1988.

[LaFi] R.E. Ladner and M.J. Fisher, Parallel Prefix Computation, *JACM,* Vol. 27, 1980, pp. 831-838.

[LiLo] R.J. Lipton and D. Lopresti, Delta Transformations to Simplify VLSI Processor Arrays for Serial Dynamic Programming, *Proceedings of the 1986 International Conference on Parallel Processing,* 1986, pp. 917-920.

[LiWa] G-J. Li and B.W. Wah, Systolic Processing for Dynamic Programming Problems, *Proceedings of the 1985 International Conference on Parallel Processing,* 1985, pp. 434-441.

[Ryt] W. Rytter, Fast Parallel Computations for Some Dynamic Programming Problems, Tech. Report, Department of Computer Science, University of Warwick, June 1987.

[Tho] M.E. Thomas, A Survey of the State of the Art in Dynamic Programming, *Amer. Inst. of Indust. Engin.,* Vol. 8, 1976, pp. 59-69.

[TsLaCh] W.W. Tsang, T.W. Lam and F.Y.L. Chin, An Optimal EREW Parallel Algorithm for Parenthesis Matching, (in preparation).

[WoCh] C.K. Wong and A.K. Chandra, Bounds for the String Editing Problem, *J. Assoc. Comp. Mach.*, Vol. 23, No. 1, 1976, pp 13-16.

[Wyl] J.C. Wyllie, The Complexity of Parallel Computations, PhD dissertation, Computer Science Department, Cornell University, Ithaca, NY, 1981.