International Journal of Pure and Applied Mathematics
Volume 18 No. 3 2005, 359-363

REMARKS ON SOME PARALLEL ALGORITHMS FOR PREFIX COMPUTATION

Ioana Chiorean
Department of Applied Mathematics
Babeş-Bolyai University
Cluj-Napoca, 400084, ROMANIA
e-mail: ioana@cs.ubbcluj.ro

Abstract: The main purpose of this paper is to make a comparison, from complexity of computation point of view, between some parallel algorithms for prefix computation.

AMS Subject Classification: 92D20, 65Y05, 68W10
Key Words: parallel algorithms, prefix computation

1. Introduction

The prefix computation has a major application in comparing two biological sequences. So, the molecule of deoxy-ribonucleic acid (DNA), which is the major carrier of genetic material in living organisms is composed of four nucleotides: adenine, guanine, cytosine and thymine, usually denoted A, G, C, T). So, a hypothetical double-stranded DNA molecule can be represented (according to [2]) as:

$$ACGTGGTAGAGGACACTAT$$
$$TGCACCATCTCTCTGGACA$$

In this sense, the biological molecules are sequences defined over an alphabet of characters and important biological problems may be reduced to operations with sequences. Many of these problems are solved by means of the deter-

Received: November 26, 2004

mination of the similarity between sequences. Here comes the role of prefix computation. Due to the fact that these sequences are very long, the problem of execution time is important. One possibility to reduce it is by improving the speed-up of execution. This can be done in different ways, but we are interested in the case of a parallel execution, it means using several processors instead of one.

The literature contains many parallel algorithms for the problem of prefix computation. In Section 3, some of them are given. Generally, they generate a logarithmic time of execution. In this paper, starting from the results obtained in [5], we study the complexity of computation both for the parallel algorithm given in [7] and for that given in [5].

2. Prefix Computation Algorithm Revisited

In order to fix the idea, we give the procedure of prefix computation.

So, given a sequence of elements $a_1, a_2, \dots, a_n$, the prefix computation is the problem of finding the vector

$$\begin{aligned}
&s_1 = a_1\,,\\
&s_2 = a_1 + a_2\,,\\
&\dots\\
&s_n = a_1 + a_2 + \cdots + a_n\,,
\end{aligned}$$

where the operation "+" can be any associative operation. Using one processor (it means the serial case) the algorithm is the following:

```
$s_0 := a_0$
for $i := 1$ to $n$ do
        $s_i := s_{i-1} + a_i$
endfor
```

Obviously, the amount of work is $O(n)$ and the computation involves only n additions.

3. Parallel Approaches for Prefix Computation

Many parallel algorithms for prefix computation exists. For instance, in [4], a parallel algorithm which needs a logarithmic time of execution is given, the processors being connected in a ring configuration (for details on ring networks, see also [3]). Using a binary tree connectivity, in [1] is given another parallel algorithm with the same performance. In [8], in a given iteration, the current value in the result array is shifted by an amount that doubles between iterations. The shifted array is added into the current array. The algorithm takes a logarithmic number of iterations to complete. Also, in [7], the following algorithm is presented:

```
{Calculate a function, $f$, on all prefixes of an $n$-element array, that is,
$s[0], f(s[0], s[1]), f(s[0], f(s[1], s[2])), \ldots, f(s[0], \ldots, f(s[n-2], s[n-1])$
$\ldots)$, using $O(n)$ processors in $O(\log n)$ time }
for $j := 0$ to $\log(n-1)$ do
        for $i := 2^j$ to $n-1$ parallel-do
            $s[i] := f(s[i-2^j], s[i])$
```

where log is the logarithm base 2, and parallel-do does the innermost computations in parallel. The function must be *associative.*

In [5], a parellel algorithm is given, based on the rewriting the relations involved in prefix computation as linear recurrence relations, in the following way:

$$\begin{aligned} &s_1 = a_1\,, \\ &s_2 = s_1 + a_2\,, \\ &\ldots \\ &s_n = s_{n-1} + a_n\,, \end{aligned}$$

with $s_0 = 0$, or, in matrix form:

$$\begin{bmatrix} s_1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & a_1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} s_0 \\ 1 \end{bmatrix}, \begin{bmatrix} s_2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & a_2 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} s_1 \\ 1 \end{bmatrix}, \ldots,$$

$$\begin{bmatrix} s_n \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & a_n \\ 0 & 1 \end{bmatrix} \begin{bmatrix} s_{n-1} \\ 1 \end{bmatrix}.$$

So, every s_i can be computed as:

$$\begin{bmatrix} s_i \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & a_i \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & a_{i-1} \\ 0 & 1 \end{bmatrix} \cdots \begin{bmatrix} 1 & a_1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} s_0 \\ 1 \end{bmatrix}.$$

The double recursive technique is applied, in this case, to compute the matrix product. As in [3], the processors are connected in a ring network and the result is obtained in $\log(i)$ time, for any $i = 1, n$.

4. The Complexity of Computation

In this section, we make some remarks concerning the effort of computation of the algorithm presented in Section 3.

The final s_n needs the computation of n matrix product. Under a binary tree network, every two processors will compute a matrix product, so in $\log(n)$ steps, the result is obtained. The execution time is, then, $\log(n) * number\_of\_operations\_involves\_in\_matrix\_computation$.

In order to evaluate the effort of computation, we need to know the number of operations involve in a two order matrix product.Taking into account that two processors work together to obtain the matrix result, they will compute, firstly, the elements of the first line and then the elements of the second line. So, the total number of operations is: two multiplications and two additions. Then, the total effort will be equal to $2\log(n) * (CP(+) + CP(*))$.

Note. As in [6], we denote by CP(.) the computational complexity of an operation.

In [7], the number of computation steps are also $\log(n)$, but the connectivity among processors is not necessarily that of a binary tree, so the effort of computation may be greater than in our algorithm. Also, the recurrsive call of function f implies an extra amount of computation.

References

[1] S. Aluru, N. Futamura, K. Mehrotra, Parallel biological sequences using prefix computation, *Technical Report* (2000).

[2] B. Berger, *Introduction to Computational Molecular Biology,* MIT Comp, Biology Ed. (1998).

[3] I. Chiorean, *Calcul Paralel,* Ed. Microinformatica, Cluj-Napoca, Romania (1995).

[4] I. Chiorean, On the complexity of some parallel algorithms for biological sequences comparison, In: *Proc. of MEDINF 2003 Int. Conference*, Craiova (2003), 117-118.

[5] I. Chiorean, I.D. Chiorean, Prefix computation in biological sequences comparison using linear recurrence relations, In: *Proc. of MEDICON 2004 Int. Conference*, Ischia, Italy (2004).

[6] Gh. Coman, D. Johnson, *Complexitatea Algoritmilor,* Litografia Universitatii Babes-Bolyai, Cluj-Napoca (1983).

[7] NIST, Internet site address: http://www.nist.gov/dads/HTML/parallelprefix.html

[8] Internet site address: http:// www.hpjawa.org/papers/HPJava/HPJava/node61.html