Computing Maximum Subsequence in Parallel*

C. E. R. Alves[1], E. N. Cáceres[2], and S. W. Song[3]

[1] Universidade São Judas Tadeu, São Paulo, SP - Brazil,
prof.carlos_r_alves@usjt.br
[2] Universidade Federal de Mato Grosso do Sul, Campo Grande, MS - Brazil,
edson@dct.ufms.br
[3] Universidade de São Paulo, São Paulo, SP - Brazil,
song@ime.usp.br

Abstract. The maximum subsequence problem finds the contiguous subsequence of n real numbers with the highest sum. This is an important problem that arises in several contexts in Computational Biology in the analysis of DNA or protein sequences. The maximum subsequence problem of n given scores can be solved sequentially in $O(n)$ time. In this paper we present an efficient BSP/CGM parallel algorithm that requires a constant number of communication rounds. In the proposed algorithm, the input is partitioned equally among the processors and the sequence stored on each processor is reduced to only five numbers. This reduction is crucial as it allows all the resulting values to be concentrated on a single processor which runs an adaptation of the sequential algorithm to obtain the result. The amount of data transmitted is $5p$ where p is the number of processors, thus independent of the input size n. The good performance of the parallel algorithm is confirmed by experimental results run on a 64-node Beowulf parallel computer, giving almost linear speedup.

Topic of interest: Algorithms and Software Tools for Computational Molecular Biology

1 Introduction

Given a sequence of real numbers, the problem of identifying the (contiguous) subsequence with the highest sum is called the *maximum subsequence problem* [2]. If the numbers are all positive, the answer is obviously the entire sequence. It becomes interesting when there are also negative numbers in the sequence.

The maximum subsequence problem arises in several contexts in Computational Biology in the analysis of DNA or protein sequences. Many such applications are presented in [6], for example, to identify transmembrane domains in proteins expressed as a sequence of amino acids. Karlin and Brendel [4] define scores ranging from -5 to 3 to each of the 20 amino acids. For the human

* Partially supported by FINEP-PRONEX-SAI Proc. No. 76.97.1022.00, FAPESP Proc. No. 1997/10982-0, CNPq Proc. No. 52.3778/96-1, 55.2028/02-9, 30.5218/03-4.

β_2-adrenergic receptor sequence, disjoint subsequences with the highest scores are obtained and these subsequences correspond to the known transmembrane domains of the receptor.

The maximum subsequence problem of n given scores can be solved sequentially in $O(n)$ [1, 2]. A variation of the maximum subsequence problem is to obtain all maximal subsequences of a sequence of n given scores. Given n scores, we can obtain the subsequence with the greatest score sum. Having obtained the kth highest score subsequence, we obtain the next $(k+1)$th highest score subsequence, disjoint from the previous k subsequences. The all maximal subsequence problem can be solved elegantly with $O(n)$ time complexity [6].

Another variation is the 2-D maximum subsequence problem, where we wish to obtain the maximum sum over all rectangular subregions of a given $n \times n$ matrix. Parallel algorithms for the 1-D and 2-D versions are presented by Wen [9] for the EREW PRAM. Both the 1-D version and 2-D version algorithms take $O(\log n)$ time using, respectively, $O(n/\log n)$ and $O(n^3/\log n)$ processors. On the other hand, Qiu and Akl [5] developed parallel algorithms for the 1-D and 2-D versions of the problem on several interconnection networks such as the hypercube, star and pancake interconnection networks of size p. The 1-D algorithm takes $O(n/p + \log p)$ time with p processors and the 2-D algorithm takes $O(\log n)$ time with $O(n^3/\log n)$ processors.

In this paper we propose an efficient parallel algorithm on the BSP/CGM computing model for the basic maximum subsequence problem. The proposed algorithm takes $O(n/p)$ parallel time with p processors and a constant number of communication rounds in which $O(p)$ numbers are transmitted. Experimental results are obtained by running the algorithm on a 64-node Beowulf parallel machine. Very promising results are presented at the end of this paper showing that the algorithm is efficient not only in theory but also in practice. To our knowledge, there are no BSP/CGM algorithms for this problem in the literature.

2 Problem Definition and the Sequential Algorithm

Consider a sequence of n real numbers or scores $(x_1, x_2, \ldots, x_n)$. A *contiguous subsequence* is any contiguous interval $(x_i, \ldots, x_j)$ of the given sequence, with $1 \leq i \leq j \leq n$. For simplicity, we use the term *subsequence* to mean *contiguous subsequence* throughout this paper. In the maximum subsequence problem we wish to determine the subsequence $M = (x_i, \ldots, x_j)$ that has the greatest total score $T_M = \sum_{k=i}^{j} x_k$. Without loss of generality, we assume at least one of the x_i is positive. With this, we have always a positive total score for the maximum subsequence problem.

Obviously if all the numbers in the sequence are positive, then the maximum subsequence is the entire original sequence. We allow the scores to be negative numbers. For instance, given the sequence $(3, 5, 10, -5, -30, 5, 7, 2, -3, 10, -7, 5)$, the maximum sequence is $M = (5, 7, 2, -3, 10)$ with total score $T_M = 21$.

There is a simple and elegant sequential algorithm of $O(n)$ for the maximum subsequence problem [1, 2]. It is based on the idea that if we have already

determined the maximum subsequence M of total score T_M of the sequence $(x_1, x_2, \ldots, x_k)$, then we can easily extend this result to determine the maximum subsequence of the sequence $(x_1, x_2, \ldots, x_k, x_{k+1})$. This is shown in Algorithm 1.

In this algorithm, we consider two cases. In the first case, x_k is the last number of the maximum subsequence M. Then if $x_{k+1} > 0$, just append x_{k+1} to M and add the value of x_{k+1} to T_M. Otherwise, M and T_M remain the same.

In the second case, x_k is not in the maximum subsequence M. Define the *maximum suffix* of the sequence $(x_1, x_2, \ldots, x_k)$ to be the suffix $S = (x_s, \ldots, x_k)$ with the maximum score T_S. In this case, steps 6 to 14 of Algorithm 1 show how to extend the result of M and the corresponding total score T_M.

Algorithm 1 Sequential Algorithm to Extend the Maximum Subsequence

Input: The maximum subsequence M of the sequence $(x_1, x_2, \ldots, x_k)$ with of total score T_M; the maximum suffix $S = (x_s, \ldots, x_k)$ with total score T_S.
Output: The updated maximum subsequence and its score for the sequence $(x_1, x_2, \ldots, x_k, x_{k+1})$.

```
 1: if x_k is the last number of M then
 2:    if x_{k+1} > 0 then
 3:       append x_{k+1} to M and set T_M := T_M + x_{k+1}
 4:    end if
 5: else
 6:    if T_S + x_{k+1} > T_M then
 7:       append x_{k+1} to S, set T_S := T_S + x_{k+1}, M := S and T_M = T_S
 8:    else
 9:       if T_S + x_{k+1} > 0 then
10:          append x_{k+1} to S and set T_S := T_S + x_{k+1}
11:       else
12:          set S to be empty
13:       end if
14:    end if
15: end if
```

For example, if the sequence $(x_1, x_2, \ldots, x_k)$ = (3, 5, 10, -5, -30, 5, 7, 2, -3, 10, -7, 5), then M is $(5, 7, 2, -3, 10)$ with score $T_M = 21$ and S is (5, 7, 2, -3, 10, -7, 5) with score $T_S = 19$. Now suppose that we want to extend the result by appending a new element to the original sequence, say $x_{k+1} = 40$. Then, by steps 6 and 7 of the algorithm, S becomes (5, 7, 2, -3, 10, -7, 5, 40) with new score $T_S = 59$, and M will be equal to S with the new score $T_M = 59$.

The sequential algorithm takes $O(n)$ time, since Algorithm 1 to extend the result when one more element is added takes constant time.

3 The Parallel Algorithm

We propose a parallel algorithm for the maximum subsequence problem for a given sequence of n scores. We use the BSP/CGM (coarse-grained multicomputer) model [3, 8], with p processors, where each processor has $O(n/p)$ local memory. This algorithm requires a constant number of communication rounds. The implementation results are shown at the end of this paper.

Consider a given sequence of n scores $(x_1, x_2, \ldots, x_n)$. Without loss of generality, we assume that n is divisible by p. They are partitioned equally into p intervals, such that each of the p processors stores one interval. Thus the interval $(x_1, \ldots, x_{n/p})$ is stored in processor 1, the interval $(x_{n/p+1}, \ldots, x_{2n/p})$ is stored in processor 2, and so on.

We now show that each interval of n/p numbers can be reduced to only five numbers.

Each processor stores n/p consecutive numbers of the input. Without loss of generality, denote the interval of n/p numbers stored in it by

$$I = (y_1, y_2, \ldots, y_{n/p}).$$

We show that it is possible to partition I into five subsequences, denoted by

$$P, N_1, M, N_2, S$$

where

1. $M = (y_a, \ldots, y_b)$ is the maximum subsequence of I, with score $T_M \geq 0$.
2. $P = (y_1, \ldots, y_r)$ is the maximum prefix of I, with score $T_P \geq 0$.
3. $S = (y_s, \ldots, y_{n/p})$ is the maximum suffix of I, with score $T_S \geq 0$.
4. N_1 is the interval between P and M, with score $T_{N_1} \leq 0$.
5. N_2 is the interval between M and S, with score $T_{N_2} \leq 0$.

Each processor finds the maximum subsequence M of I, the maximum prefix P of I and the maximum suffix S of I.

We have several cases to consider.

If all the y_i are negative numbers, then we assume M, P and S empty with $T_M = T_P = T_S = 0$, N_1 is the entire I and N_2 empty with $T_{N_2} = 0$.

We now show that

Lemma 1. *If M is not empty, then one of the following cases must hold.*

1. *P is to the left of M, with $r < a$, and with N_1 in between.*
2. *M is equal to P, with $a = 1$ and $b = r$. We have no N_1.*
3. *M is a proper subsequence of P, with $a > 1$ and $b = r$. We have no N_1.*

Proof. If $r < a$, case 1 holds.

Let us suppose that $r \geq a$. We have to prove that $r = b$, showing that 2 or 3 holds.

With $r \geq a$, if $r < b$ then the score of $(y_a, \ldots, y_r)$ is smaller than T_M, so the score of $(y_{r+1}, \ldots, y_b)$ is positive. Then the prefix $(y_1, \ldots, y_b)$ would have a score greater than T_P, a contradiction.

Similarly, with $r \geq a$ and $b < r$, $(y_{b+1}, \ldots, y_r)$ would have a positive score and $(y_a, \ldots, y_r)$ would have a score greater than T_M, again a contradiction. So $r \geq a$ leads to $r = b$.

We have also the following lemma regarding the maximum suffix S with a similar proof.

Lemma 2. *If M is not empty, then one of the following cases must hold.*

1. *S is to the right of M, with $s > b$, and with N_2 in between.*
2. *M is equal to S, with $a = s$ and $b = n/p$. We have no N_2.*
3. *M is a proper subset of S, with $a = s$ and $b < n/p$. We have no N_2.*

The five values T_P, T_{N_1}, T_M, T_{N_2} and T_S for each interval are used in the parallel algorithm. When M and P are not disjoint, that is, M is a subsequence of P, whether proper or not, we redefine T_P to be 0 and T_{N_1} to be the non-positive score of the prefix that immediately precedes M. A similar adaptation in done with S and T_{N_2} when M and S are not disjoint. It is easy to see that after this redefinition,

$$T_P + T_{N_1} + T_M + T_{N_2} + T_S = \sum_{i=1}^{n/p} y_i,$$

$$\text{score of } P = \max\{T_P,\ T_P + T_{N_1} + T_M\},\ \text{and}$$

$$\text{score of } S = \max\{T_M + T_{N_2} + T_S,\ T_S\}.$$

Thus, in this way, we build a sequence of five numbers with the same scores as in the original interval, regarding the total score (entire sequence), maximum subsequence, maximum prefix and maximum suffix. The seemingly useless zeros are kept to simplify the bookkeeping in the last step of the parallel algorithm.

Having computed the five numbers mentioned above, each processor sends them to processor 1. Processor 1 solves the maximum subsequence problem of the $5p$ numbers sequentially, in $O(p)$ time and reports the solution.

We now present the complete parallel algorithm.

Theorem 1. *Algorithm 2 correctly computes the maximum subsequence of $(x_1, x_2, \ldots, x_n)$ in a constant number of communication rounds involving the transmission of $O(p)$ numbers and $O(n/p)$ local computation time.*

Proof. The correctness of the parallel algorithm is based on Lemma 1 and Lemma 2. Also, it is easy to see that the maximum subsequence considering the $5p$ values corresponds to the maximum subsequence of the original sequence. If the latter is entirely contained in one of the p intervals, the correspondence is

Algorithm 2 Parallel Maximum Subsequence

Input: The input sequence of n numbers $(x_1, x_2, \ldots, x_n)$ equally partitioned among the p processors.

Output: The maximum subsequence of the input sequence.

1: Let the sequence stored in each processor be $I = (y_1, y_2, \ldots, y_{n/p})$. Each processor obtains the maximum subsequence M of I with score T_M.
2: Each processor obtains the maximum prefix P with score T_P, and obtains the maximum suffix S with score T_S. The interval between P and M is N_1 with score T_{N_1}; the interval between M and S is N_2 with score T_{N_2}.
3: Consider Lemma 2 and redefine the appropriate values of T_P, T_{N_1}, T_M, T_{N_2}, T_S if necessary.
4: Each processor sends the five values T_P, T_{N_1}, T_M, T_{N_2}, T_S to Processor 1.
5: Processor 1 receives the $5p$ values and computes the maximum subsequence of the received values.
6: Let the maximum subsequence obtained be $m_1, \ldots, m_k$. The processor that stores m_1 can easily compute the start index of the maximum subsequence correponding to the original input, while the processor that stores m_k can compute the end index of the anwer.

direct. Otherwise, it starts within an interval (being its maximum suffix), spans zero or more entire intervals, and ends within another interval (being its maximum prefix). The $5p$ values contain all the necessary information to find this subsequence. In step 2, we need one communication round in which each processor sends five values. In step 6, processor 1 needs some information from the processor to compute the start and finish indices of the maximum sequence. So we need another communication round. In step 2, to obtain P, N_1, M, N_2 and S, each processor runs an algorithm carefully adapted from the well-known sequential algorithm of [1, 2], in such a way that all the five values can be obtained by scanning the n/p numbers only once. In step 5, processor 1 runs an $O(p)$ algorithm. Thus, the local computation time is $O(n/p)$, given that $n/p > p$, a common assumption of the BSP/CGM model.

4 Experimental Results

We have run the parallel algorithm on a 64-node Beowulf machine consisting of low cost microcomputers with 256MB RAM, 256MB swap memory, CPU Intel Pentium III 448.956 MHz, 512KB cache. The cluster is divided into two blocks of 32 nodes each. The nodes of each block are connected through a 100 Mb fast-Ethernet switch. Our code is written in standard ANSI C using the LAM-MPI library. We assumed an input size of $n = 1,830,000$ [4] and used randomly generated data. Figure 1 shows the total running times (computation plus communication), Figure 2 the communication times, and Figure 3 the speedups obtained.

[4] This size corresponds to the number of nucleotide pairs of the bacterium *Haemophilus influenzae*, the first free-living organism to have its entire genome sequenced [7].

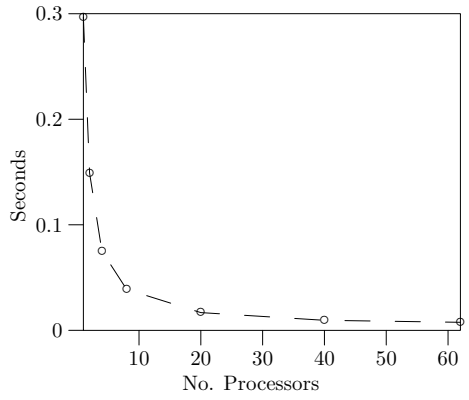

Fig. 1. Total times (computation + communication) for input size n=1,830,000.

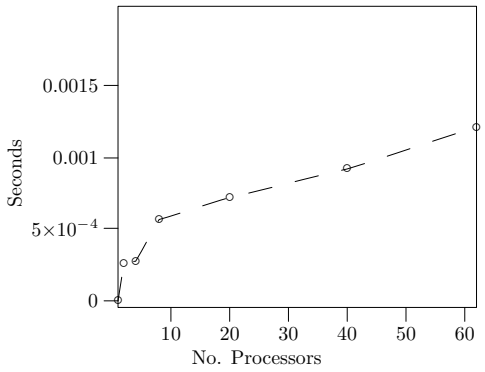

Fig. 2. Communication times for input size n=1,830,000.

5 Conclusion

We propose an efficient parallel solution to the maximum subsequence problem that finds the contiguous subsequence of n real numbers with the greatest total score, an important problem in the analysis of DNA or protein sequences to identify subsequences with desired properties. In the proposed algorithm, the input is partitioned equally among the processors and the sequence stored on each processor is reduced to only five numbers. This reduction is crucial as it allows all the resulting values to be concentrated on a single processor which runs an adaptation of the sequential algorithm to obtain the result. The amount of data transmitted is $5p$ where p is the number of processors, thus independent of the input size n. Our algorithm not only finds the maximum score of the subsequence, but also the subsequence proper. The good performance of the parallel algorithm is confirmed by experimental results run on a 64-node Beowulf parallel computer, giving almost linear speedup. Finally we must say that the

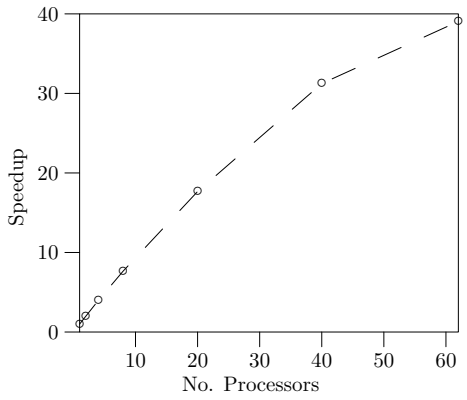

Fig. 3. Speedups for input size n=1,830,000.

sequential algorithm is very efficient. The parallel version is only justified for large sequences.

Acknowledgments

We would like to thank the anonymous referees for their review and helpful comments.

References

1. J. L. Bates and R. L. Constable. Proofs as programs. *ACM Transactions on Programming Languages and Systems*, 7(1):113–136, January 1985.
2. J. Bentley. *Programming Pearls*. Addison-Wesley, 1986.
3. F. Dehne, A. Fabri, and A. Rau-Chaplin. Scalable parallel geometric algorithms for coarse grained multicomputers. In *Proc. ACM 9th Annual Computational Geometry*, pages 298–307, 1993.
4. S. Karlin and V. Brendel. Chance and significance in protein and dna sequence analysis. *Science*, 257:39–49, 1992.
5. K. Qiu and S. G. Akl. Parallel maximum sum algorithms on interconnection networks. Technical report, Queen's Unversity, Department of Computer and Information Science, 1999. No. 99-431.
6. W. L. Ruzzo and M. Tompa. A linear time algorithm for finding all maximal scoring subsequences. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, pages 234–241. AAAI Press, August 1999.
7. D. P. Snustad and M. J. Simmons. *Principles of Genetics*. John Wiley and Sons, 2000.
8. L. Valiant. A bridging model for parallel computation. *Communication of the ACM*, 33(8):103–111, 1990.
9. Zhaofang Wen. Fast parallel algorithm for the maximum sum problem. *Parallel Computing*, 21:461–466, 1995.