Parallel Processing Letters Vol. 3 No. 3 (1993) 243–252

PRAS: PARALLEL ALIGNMENT OF SEQUENCES ALGORITHM

RAJENDRA KULKARNI, SHASHANK DATE and BHAVNA KULKARNI
Centre For Development of Advanced Computing
Pune University Campus, Ganeshkhind Road, Pune, 411 007, India

URMILA KULKARNI and A. S. KOLASKAR
BIOINFORMATICS, Department of Zoology
University of Pune, Pune 411 007, India

Received 28 September 1992
Revised 14 May 1993
Accepted by P. C. P. Bhatkar

ABSTRACT

This paper presents an algorithm for multiple alignment of sequences of proteins and nucleic acids on loosely-coupled MIMD machines. An inherently parallel algorithm is devised and implemented on a transputer-based multiprocessor platform. The results indicate the possibility of scalable performance.

Keywords: Multiple alignment, transputer, dynamic programming, farming.

1. Introduction

The objective of this paper is to present details of an algorithm of the multiple alignment problem which is one of the most important problems in molecular biology. It has applications in various fields like genetic engineering, drug design, etc. It is a very challenging problem for computational sciences as well. In this paper, we present the problem as a core computatioal problem related to strings. It can be described as follows.

Given three strings "ABC", "BC", "AC" and the freedom to insert gaps as characters having no meaning (or freedom to slide the string over each other), what is the way to align them so that the number of identities in the strings are maximised?

For such a small problem, the answer is intuitively obvious. The strings can be arranged in the following fashion.

```
A B C
- B C
A - C
```

However, this trivial-looking problem becomes extremely intractable as the strings grow large and the number of strings to be aligned increases. It is an NP-complete problem.

In biology, this problem has interesting implications. The strings mentioned above become the primary structure information of the sequences of proteins or nucleic acids, and alignment of a particular part of sequences becomes an important clue to predict the structure and function relationship. The problem is further complicated by the fact that some proteins/nucleic acids can fully or partially replace each other. This fact needs to be incorporated in the problem. Then the problem becomes not to find out an optimal alignment that maximises the number of identities, but that of finding the maximal alignment score after considering the substitutions also.

So, the problem described above has got an additional consideration of substitution. Suppose A and B can be freely substituted. Then the resulting alignment would be as given below.

A B C
B - C
A - C

We have introduced a new concept of substitution. This is abstracted in form of matrix called a substitution matrix which looks something like this:

A	B		C
A	1	1	-
B	1	1	-
C	-	-	1

This matrix simply indicates that if you substitute A with B, you get the identical score. 'Score' is a kind of return function that one gets by substitution of one letter with another. The multiple alignment problem now becomes that of maximising the score. In the matrix mentioned above, only 0 and 1 values were there. In a real biological case, there are integer valued matrices. These matrices are called Unitary protein, Mutation Data matrix, etc. These matrices encapsulate the research findings of the biologists.

The final consideration in the problem is that of penalty. So far we have considered that strings can be freely broken for insertion of gaps. In the physical world, it is not so. The molecules cannot be that freely broken down, therefore, there is some penalty associated with the gap. This represents a kind of barrier value, and only if the benefit from the insertion of gaps exceeds the barrier value, is the insertion of gaps accepted.

Now the problem can be stated as follows:

Find the optimal alignment of the given sequences, the substitution matrices and penalty values for insertion of gaps.

2. Literature Review

The basic contribution towards the solution of this problem has been from Needleman and Wunsch [1]. They formulated the pairwise alignment problem as a dynamic programming problem. There are many heuristic methods [2].

The major problem in these methods have been that these methods are either fast or accurate. The quality and optimality of alignment and the computational time have a trade-off. Kulkarni *et al.* [3] considered a different approach. It is accepted that the problem cannot be solved in a monolithic transformation that maps the feasible solution set to the optimal solution set in a single transformation. It is computationally involved and the convergence towards optimality is not guaranteed. Therefore, the decomposition approach was resorted to. The decomposition approach, by definition, is the transformation of the problem into a series of subproblems that are coupled to each other and then synthesizing the optimal solution from the solution of subproblems. It has various advantages like reduction in dimensions of the problem, simplicity in computation and better understanding of the problem.

In this paper, we present the details of the algorithm for the multiple alignment problem, and present results of parallelisation to show that there is a lot of potential for parallelisation in this problem. The details of program optimisation are also presented.

3. Multiple Alignment Algorithm

Description: The algorithm consists of three steps.

1. Pairwise alignment step: A dynamic programming algorithm is devised for pairwise alignment. It is a variant of the Needleman and Wunsch algorithm. Every possible pair combination of the sequences is aligned and corresponding alignment scores are calculated.
2. Clustering step: Based upon the alignment scores, the sequences are clustered together using the single linkage clustering method.
3. Intergroup profiling step: Sequences in different clusters are aligned together by using a different scoring matrix based on the position of a character at a particular place. It is again a dynamic programming problem.

The rationale behind the development of this algorithm can be explained as follows:

In the first step, one finds pairwise similarity scores and these scores allow groups of similar sequences to be clustered together. In the next step, similar sequences in a cluster are aligned together so that instead of a pair of sequences, multiple sequences with a lot of similarity amongst them are aligned together. The last step consists of aligning different clusters of sequences together while treating each individual group of sequences as already aligned sequence. It is envisaged that the process of gradually bringing together the dissimilar sequences will give good quality of alignment.

3.1. Pairwise Alignment Step

Mathematical formulation of the pairwise alignment problem is presented below.

Let,

A, B are the strings being aligned,
a = length of string A,
b = length of string B,
i = position index,
j = position index,
k = position index,
S = scoring matrix of dimensions a X b,
$S[i, j]$ = result of substitution of ith *character* in sequence a by jth *character* in sequence b which constitutes net return function for dynamic programming formulation,
$M[i, j]$ = i, jth element of M which serves as a cumulative return function for dynamic programming problem,
M = final score matrix of dimensions b X a, for base run 0 cumulative return function,
A = final alignment score,
n = run number,
M_n = matrix for nth run,
M_n^{max} = Maximum score in the last row and last column of M_n matrix,
N = number of runs,
sd = standard deviation of the scores of N runs.
penalty = penalty for introducing a gap in any sequence.

The pairwise alignment problem is a problem of 'a' stages and 'b' states. At every stage-state combination, the decision is whether to insert a gap or to match the given character.

The recursive equation for this problem is presented below, this is to be solved for $i = 2, \ldots,$ a stages and $j = 2, \ldots,$ b states.

$$M[i, j] = Max \begin{cases} M[i+1, j+1] & \\ M[k, j+1] + & penalty \quad for\ k = i+2, \ldots, b; \\ M[i+1, k] + & penalty \quad for\ k = j+2, \ldots, a. \end{cases} \tag{1}$$

With the initial condition that:

$$M[l, a] = S[k, a] \quad for\ k = 1, \ldots, b \tag{2}$$

$$M[b, l] = S[b, k] \quad for\ k = 1, \ldots, a. \tag{3}$$

It will be evident that the pairwise alignment problem is treated as a multistage decision process with the following decisions taken at every stage and state:

Either keep the characters in both the sequences the same, or insert a gap in one of the sequences to improve the number of identities score.

In the ideal condition, the decision space of the problem is infinite, in a sense that an infinite number of gaps can be inserted at every place to find out whether it results into a better alignment. However, it is not possible, hence only one stage lookback is provided as mentioned above. The algorithm rests on the assumption

that no matter what the previous stage and state might be, an optimal decision is always the one that is optimal for the current stage and state (Bellman's principle of optimality).

To improve upon the score M_0^{max} calculated in the above calculation, the sequences are randomly shuffled and then scores are calculated. This is done for N number of times. The shuffle ensures that at least a or b positions in the sequences are changed per shuffle. The idea is to find out whether the final alignment score is the result of applying the algorithm or simply of a random chance. The normalised alignment score is calculated as follows:

$$A = \frac{M_0^{max} - \frac{\sum_{i=0}^{n} M_i^{max}}{N}}{sd}. \tag{4}$$

3.2. Clustering Step

The alignment scores are calculated for all possible pairs of sequences and are used to cluster them together using single linkage cluster analysis. It is a well-known method and hence the details of the method are not presented here. It may be noted that if U sequences are to be aligned then U * (U-1)/2 scores are to be calculated.

3.3. Inter Group Profiling Step

In this step the sequences in one cluster are treated as a single entity and these groups of sequences are aligned against each other. The details of the algorithm are presented below.

Suppose we are trying to align two groups, group 'a' and group 'b'.

Let,

A = two-dimensional vector of group a, consisiting of sequences in group a arranged one over another in K rows, a typical letter A is denoted by the $A[k, i]$ which means ith letter of k kth sequence,

B = two-dimensional vector of group b, consisiting of sequences in group b arranged one over another in K rows,

k, i, j = position index,

K = number of sequences in group a,

L = number of sequences in group b,

S^* = substitution matrix containing a score of the result of substitution of ith element with jth element,

S = original substitution matrix consisting of a typical element $S[i, j]$ representing the score of replacement of letter i by letter j.

The new substitution matrix is calculated as follows:

$$S^*[i, j] = \sum_{k=1}^{K} \sum_{l=1}^{L} Pr(A[k, i]) * Pr(B[l, j]) * S_{[A[k,i], B[l,j]]}. \tag{5}$$

The probablity calculation in the above matrix is done as follows:

$Pr(A[k, i])$ = frequency of occurrence of letter $A[k, i]$ in K rows divided by K, that is, the number of sequences in group a.

With this new substitution matrix again the dynamic programming problem mentioned in the equation is solved. The only difference between the previous problem and this is: the decision regarding the insertion of gaps is operated on the entire group of sequences and the appearance of a letter in a sequence at a particular position is weighted equally with its occurrence at the same position along with other similar sequences.

This method is similar to the one used by Vingron and Argoss [2] in their profiling algorithm.

4. Parallel Processing Techniques

There are two basic possibilities of parallelisation in this algorithm, namely, data parallelisation and the other is algorithmic parallelisation. The data parallelisation scheme is used in this algorithm.

- Pairwise alignment step. The data parallelisation is in the first step of the algorithm. Each pair of the sequences is treated as a seperate work unit and is given to different processors. Thus, the calculation of pairwise scores is done in parallel. The result that comes out is the alignment score.
- Profiling step. As it is obvious from the algorithm, each element of the profile matrix requires two vectors $A[k, i]$, $k = 1, \ldots, K$ and $B[l, j]$, $l = 1, \ldots, L$, i.e. each element calculation in the profile matrix requires two segments of the given sequences. In addition, it requires an original substitution matrix. Therefore, the calculation of each element in the profile matrix is treated as an independent work unit and the calculation of each of the elements is distributed across the processors. However, this scheme does not work out to be very effective because there is more communication than computation. Therefore, each work packet size is increased to some n element calculation where n can be changed according to the machine configuration.

5. Hardware Platform

This algorithm is developed and tested on the PARAM supercomputer developed by C-DAC, Center for Development of Advanced Computing, Pune, India. It is built around T-805 transputer with 25 MHz cycle-time as a processor. The machine has 256 nodes of such processors. It can be connected to any front end like IBM-PC, Vax, etc.

5.1. Implementation Details

The machine can be configured for any user-defined topology. The algorithm is implemented using the farming approach. In this approach, the farmer distributes

the work packet to the free abstraction of processor farms. Farming has the following advantages.

- The details of the topology are transparent to the user.
- Scalability from 1–64 nodes does not call for any change in the executable file.
- The worker tasks in both the steps of parallelisation are identical. In the first step of the algorithm, it is the calculation of alignment score and in the second case, it is the calculation of the profile matrix element. Flood-fill type of topology is ideally suitable when the worker tasks are identical. We have the results to show that processor utilisation is good with the processor farms.

6. Experimental Results

There are two phases in which experimental results are prepared. First, a few sequences of average length 140 and having known similarity are tested on Micro-Vax-II and benchmarking results are reported. The bigger-sized problems having more sequence lengths and more numbers of sequences are tested independently because the estimated time for solving these problems on Micro-Vax is very large. Table 1 presents basic benchmarking results. These are with 2 transputers and 4 transputers respectively. It is evident from these results that it will be impracticable to have these analyses done on a single processor machine.

Table 1. Comparative timings for small problems (Timings in hours:minutes:seconds format).

Number of sequences	Micro-Vax-II	2 transputers 4Mb/node	4 transputers 4Mb/node
2	0:04:12	0:01:49	0:01:01
3	0:11:35	0:03:42	0:01:10
4	0:22:00	0:05:40	0:03:10
5	0:35:00	0:08:40	0:04:50
6	0:47:00	0:13:00	0:06:30

It is seen that the timings are extremely sensitive to the number of sequences. This is obvious from the design of the algorithm. The identical readings from the first run indicate that in 4 processor cases it was not utilised properly because the root transputer tended to take more work — a typical disadvantage of the processor farms as mentioned above.

Table 2 presents the results of a whole family of cytochrome sequences aligned together (number of sequences 73, average length of the sequences 70–100). These types of analyses are not available in the literature, therefore, there are no benchmarks against which these results could be compared to.

Table 2. Results of cytochrome family alignment (Number of sequences 70, length 70–80).

Number of processors	Pairwise alignment time (in seconds)	Profiling time (in seconds)	Total time (in seconds)
64	1262.83	851.55	2114.38
32	2412.01	1245.86	3657.87
16	4799.89	2044.59	6844.59
8	9850.07	4358.56	14208.63
4	19447.58	9833.89	29330.67

The following conclusions can be drawn from this analysis:

- In the run timings, the time gains with respect to the number of processors is almost linear, which is expected because of data parallelisation and the compute intensive nature of the job.
- The time gains with respect to the number of processors in the profiling case is sublinear. It is due to various factors such as less compute intensive work packet, varying length of the sequences and more communication overhead as compared to the previous step.
- If we consider efficiency as a ratio of the following two quantities: ratio of timings and ratio of processors, then it will be evident from the table that the efficieny comes in the range of 85%, which is sublinear but reasonable considering the nature of the problem.

7. Sensitivity Analysis

The purpose of carrying out sensitivity analysis is to find out the model parameters that the given method is particulary sensitive to. For this purpose, obvious choices are the length of the sequences, the number of sequences and the nature of similarity in the sequences.

7.1. Sensitivity With Respect to the Length of Sequences

Table 3 indicates results of sensitivity analyses with respect to the length of the sequences. By sequences we mean the average length of sequences in the example problem. It is seen that this method is extremely sensitive to the length of the sequences. It is expected because in the pairwise alignment problem, length is an indicator of stages and states, and it is a known fact that time increases exponentially with the increase in the number of stages in the type of algorithm explained above. Also, in the profiling method, the size of profiling matrix is determined by the length of the sequence.

7.2. Sensitivity With the Number of Sequences

The method is sensitive to the number of sequences in a sense that time increases with the increase in the number of sequences. However, the increase or decrease in time is not combinatorially exploding (see Table 2).

7.3. Sensitivity With Respect to the Nature of Similarity Between Sequences

It is observed that this method is a powerful tool for finding out dissimilarities also. The effect of having dissimilar sequences in the alignment is that the effective length of sequences increases and hence, computational load on the profiling stage increases. As a result, computational time increases. This is evident from the fact in Table 3 run 3. In this run, sequences with known dissimilarity were aligned and, in the end, it was found that the average sequence length in the aligned sequence exceeded 400 characters.

Table 3. Sensitivity with respect to length of sequence.

Length of sequences (no. of characters)	Time seconds
73	1262.83
140	5847.08
250	28354.03

8. Summary And Concluding Remarks

In this paper, we described the multiple alignment algorithm in detail. The scope for parallelising the algorithm was indicated and it was shown that parallel processing machines can be used to tackle these problems effectively. Some experimental results were presented and the parameters to which the algorithm is sensitive for computational time was highlighted. The following conclusions can be drawn from the paper:

- The algorithm provides for guaranteed scalability of performance with increase in the number of processors.
- The algorithm is amenable to efficient parallelisation.
- The algorithm is ideally suited for processor farms.

However, this algorithm suffers from the following limitations:

- Improper load balancing: The work packet for pairwise alignment consists of two sequences and appropriate scoring matrices, and one work packet is assumed to be over when the alignment score for this pair is calculated. In such a case, there is a possibility of one pair requiring less time than the other. This limitation will be accentuated only when the sequences in a pair are very dissimilar in length. It may be argued that since the algorithm is intended to find out similarity, such dissimilar sequences will not be compared.

- The problem can be formulated as a well-posed, multiple objective, optimisation problem, but in this paper a heuristic approach is resorted to.
- The profiling method described in this algorithm does not give optimal processor utilisation. More optimisation of code is necessary.

In conclusion, it can be said that multiple alignment is an important problem not only from the biological point of view but for computational sciences too. A new implementation was presented in this paper.

References

[1] S. B. Needleman and C. D. Wunsch, A general method applicable in amino acid sequences of two proteins, *Journal of Molecular Biology* **48** (1970) 443–453.

[2] A. Vingron and P. Argoss, Fast and sensitive multiple sequence algorithm, *CABIOS* **5** (1989) 115–121.

[3] R. V. Kulkarni, S. Date, B. Kulkarni, U. Kulkarni and A. S. Kolaskar, PRAS: Parallel alignment of sequences package, *Advanced Computing*, Ed. Vijay Bhatkar *et al.* (Tata McGraw-Hill Press, 1991) pp. 392–399.