PARALLEL SEQUENCE ALIGNMENT ALGORITHM FOR CLUSTERING SYSTEM

Yang Chen, Songnian Yu, Ming Leng
School of Computer Engineering and Science, Shanghai University, Shanghai, PRC. Emai: chenyang2000@163.com

Abstract: Sequence alignment is one of the most important fundamental operations in bioinformatics. It has been successfully applied to predict the function, structure and evolution of biological sequences. In this paper, the sequence alignment algorithms based on dynamic programming are analyzed and compared. We present a parallel algorithm for pairwise alignment and implement it on a clustering system with MPI. The experimental results demonstrate the effectiveness in performance promotion. We encapsulate the algorithm into a grid service for practical use.

Key words: Bioinformatics; sequence alignment; parallel algorithm; clustering system; MPI

1. INTRODUCTION

With the genome project being carried out, new biological molecular information has emerged in large numbers, in which some valuable knowledge does exists. However, it is really hard for us to discover them. To fulfill such a demanding task, a new cross-disciplinary subject called bioinformatics has been rapidly developed.

In bioinformatics, the similarity comparison among biological sequences called *sequence alignment* is a fundamental method of information processing, which plays a significant role in revealing the function, structure and evolution of biological sequences. Sequence alignment finds out the maximal matched bases or residues by means of some specific mathematical model or algorithm whose outcome to some extent tells the relationship

Please use the following format when citing this chapter:
Chen, Yang, Yu, Songnian, Leng, Ming, 2006, in International Federation for Information Processing (IFIP), Volume 207, Knowledge Enterprise: Intelligent Strategies In Product Design, Manufacturing, and Management, eds. K. Wang, Kovacs G., Wozny M., Fang M., (Boston: Springer), pp. 311-321.

among sequences and their biological characteristics. Therefore, the design of a reasonable and efficient algorithm for sequence alignment has become a very important research issue in this field.

For the alignment of sequence pairs whose lengths are n, there are totally

$$\binom{2n}{n} = \frac{(2n)!}{(n!)(n!)} \cong \frac{2^{2n}}{\sqrt{\pi n}}$$

different cases to be considered. That seems a computational problem with exponential complexity.

Needleman and Wunsch [10] presented the first pairwise sequence alignment algorithm based on dynamic programming (DP), on which Gotoh [9] made some further modifications. An algorithm with linear space complexity was proposed by Hirschberg [6], which was later improved by Mayers and Miller [7]. Still based on DP technique, Smith and Waterman [11] presented an algorithm for local alignments.

2. BIOLOGICAL SEQUENCE ALIGNMENT ALGORITHM

The data of sequence are partitioned into DNA sequences and protein sequences. Each DNA sequence consists of four types of base A, T, C and G and each protein sequence is made up of 20 types of amino acid, hence any sequence can be represented as a string over specific alphabet. The similarity of sequences is described in quantitative values or qualitative description. The *degree of similarity* and the *edit distance* are used to quantify the similarity of sequences. The *dot plot* indicates in an intuitive and clear way the area where conspicuous similarity exists between two sequences, and then helps to locate the possible alignment.

2.1 Sequence alignment

Given a *score function*, the sequence alignment computes the optimal alignment among two or more sequences, ending up with the alignment of maximal similarity among sequences via matching the corresponding symbols or inserting gaps for insertion or deletion within sequences. Score function is a key parameter involved in sequence alignment algorithms, which is used to evaluate the *edit operations* applied to sequences (shown in Fig. 1).

$$p(a,a) = 1$$
$$p(a,b) = 0 \qquad (a \neq b)$$
$$p(a,-) = p(-,b) = -1$$

Figure 1. Evaluate edit operation with score function

Several algorithms are presented for sequence alignment. Most of them are based on dynamic programming, upon which various improvements are made. According to the number of sequences to be aligned, the algorithms are classified into pairwise alignment algorithms and multiple alignment algorithms. In this work, we mainly focus on the former ones.

There are two main criteria for the performance evaluation of sequence alignment algorithm. One is the running speed of algorithm; the other is the result sensitivity for optimal alignment. We care more about the execution time of algorithm which reflects the running speed.

2.2 Pairwise sequence alignment algorithm

The typical algorithm for global alignment is the Needleman-Wunsch algorithm, which is designed for sequence pairs with a high degree of similarity on the global level. The degree of similarity for sequence pairs is computed using an iterative method (shown in Fig. 2) and stored in the *score matrix* from which the optimal alignment is found through the backtracking procedure based on dynamic programming.

$$d_{i,j} = \max\begin{cases} d_{i-1,j-1} + p(s_i, t_j) \\ d_{i-1,j} + p(s_i, -) \\ d_{i,j-1} + p(-, t_j) \end{cases}$$

Figure 2. Compute degree of similarity iteratively

The basic algorithm for local alignment is the Smith-Waterman algorithm. Different from the global one, the local alignment algorithm is applied to sequence pairs that are distantly related with some similarity in local areas rather than on the whole. With regard to this, some slight modifications are made so as to seek the subsequence pairs with maximum similarity.

The complexity of algorithms mentioned above is determined by the scale of score matrix; in other words, it is closely related to the product of sequence lengths. The time complexity of computing score matrix is $O(n^2)$

and that of the backtracking procedure is $O(n)$, where n is the length of sequence. Therefore, the overall time complexity comes to $O(n^2)$.

2.3 Visualized implementation of algorithms

For the convenience of comparison among various sequence alignment algorithms, we implement and join them into a same graphical user interface. The program is designed based on MFC programming technology, running on Windows platform as shown in Fig. 3.

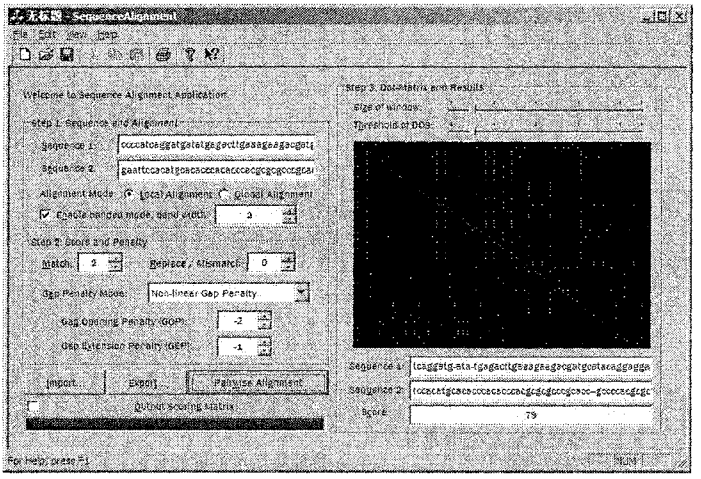

Figure 3. Visualized implementation of sequence alignment algorithms

Here, different cases of pairwise sequence alignment are integrated and the environmental parameters such as alignment mode, score function, gap penalty, etc. can be configured and adjusted freely. Meanwhile, the visualized results including score, dot-plot and optimal alignment can be instantly observed from the window.

3. PARALLEL SEQUENCE ALIGNMENT ALGORITHM

3.1 Feasibility analysis

As previously stated, the time consumption of sequential algorithm mainly lies in the computation of score matrix. The recursive formula of

score matrix indicates that the computation of $d(i, j)$ can be started only if $d(i\text{-}1, j\text{-}1)$, $d(i\text{-}1, j)$ and $d(i, j\text{-}1)$ obtain their values, which is shown in Fig. 4.

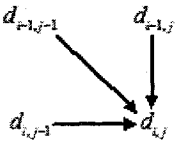

Figure 4. Data dependency in computing elements of score matrix

As a result, the computation of score matrix can be conducted successively in order of anti-diagonals. Therefore, the elements in same anti-diagonal marked by the symbol → can be computed simultaneously (Fig. 5).

		A	T	G	C
	0	1	2	3	→
A	1	0	1	→	
T	2	1	→		
G	3	→			
C	→				

Figure 5. Compute score matrix in a parallel manner

3.2 Model for problem solving

Driven by the data dependency in computing score matrix, we parallelize the pairwise alignment algorithm to improve its performance for multi-processor systems. The major models for parallel solving are listed below.

Pipeline model: As a basic unit, each row of the score matrix is computed sequentially by a processor, which blocks itself till the required elements in the row above are computed. This forms a continuous pipeline (Fig. 6).

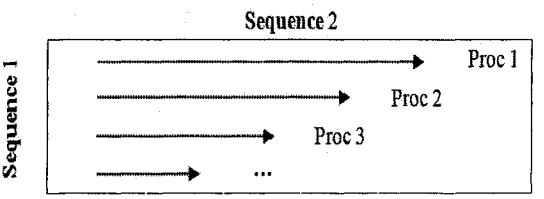

Figure 6. Pipeline model for parallel solving

Anti-diagonal model: All processors compute simultaneously along an anti-diagonal of score matrix, from the left-top corner to the right-bottom corner. Each idle processor selects an element that is not computed from current anti-diagonal. When all elements in current anti-diagonal are processed, the computation moves on to next anti-diagonal (Fig. 7).

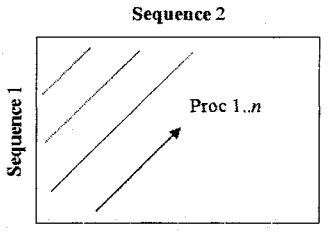

Figure 7. Anti-diagonal model for parallel solving

3.3 Design and implementation

In accord with the pipeline model above, we design a synchronous medium-grained parallel algorithm for distributed memory systems, by using single program multiple data (SPMD) technology.

The algorithm is based on the wave-front method. The score matrix is partitioned into several *bands* by row and several *blocks* by column. All the bands are distributed to multiple processors via a balanced allocation. A typical example is shown in Fig. 8. In line with the order of anti-diagonals, each processor computes the block in its own band concurrently. Due to the data dependencies, the communication between processors is required to transfer data on the boundary of bands. The height of bands and the width of blocks can affect the performance of the parallel algorithm.

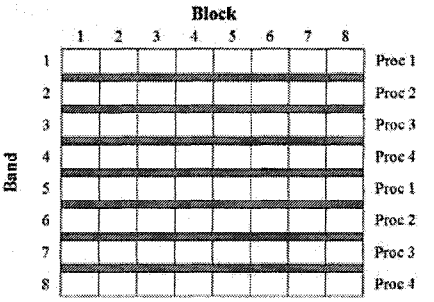

Figure 8. A 8*8 partition for 4 processors

By applying this algorithm, the time complexity is reduced to $O(n)$ when n processors are used. Specifically, if the sequence pairs to be aligned are closely related, only parts of DP matrix are worth computing. In this case, while the sequential algorithm has $O(cn)$ time complexity, the parallel counterpart can compute in $O(n)$ time with $O(c)$ processors, where c is a constant factor of restriction.

The algorithm is implemented on a clustering system with message passing interface (MPI) and it works properly on Linux platform. The outputs of algorithm for different length-specific inputs are verified, which to a certain extent guarantees the correctness of algorithm.

4. EXPERIMENTAL RESULTS

To evaluate the performance of the parallel algorithm, we design a experimental scheme to measure the execution time of algorithm under various lengths of sequences and various numbers of processors which are related to the complexity of algorithm.

The biological sequences we use for experiment are retrieved from the SRS database provided by research centre of bioinformatics, Peking University. The testing environment is the laboratory of high performance computing in Shanghai University, which is a PC cluster constituted by 70 nodes (Intel Pentium 4 CPU 2.4 GHz with 256 MB memory) running on Linux operating system (Red Hat Enterprise Linux WS 3, gcc 3.2.3, MPICH 1.2.5). The experimental results are shown in Table 1 as follows.

Table 1. Comparison in execution time of algorithms

sequence length	sequential algorithm	parallel algorithm, *np* processors					
		np = 1	*np* = 2	*np* = 4	*np* = 8	*np* = 16	*np* = 32
500	0.244804	0.365445	0.247524	0.165411	**0.126663**	0.570836	1.30005
1000	1.79986	2.71574	1.59836	0.959565	**0.654181**	1.14644	1.53744
1500	5.9714	8.83867	4.86467	2.95273	1.8195	**1.78117**	2.14013
2000	13.9534	20.3237	10.9388	6.30774	3.79232	**3.22145**	3.47226
2500	26.2785	39.5335	21.014	11.6055	6.9633	5.1999	**4.56656**
3000	45.5543	67.2282	35.4311	19.5344	11.4641	8.196	**6.54519**
3500	71.6498	106.583	55.7057	30.1991	17.4998	11.809	**8.95839**
4000	107.255	158.605	82.4512	44.275	25.0185	16.3415	**12.1245**
4500	152.542	225.746	118.905	62.4093	34.7657	21.5546	**15.9286**
5000	208.827	310.181	158.51	86.4587	46.905	28.465	**20.2812**

On one hand, given specific number of processors, the execution time of sequential algorithm and parallel algorithm with only one processor (np = 1) rises sharply with the increase of sequence length, which is shown in Fig. 9. It is highly imperative to take the advantage of parallel processing.

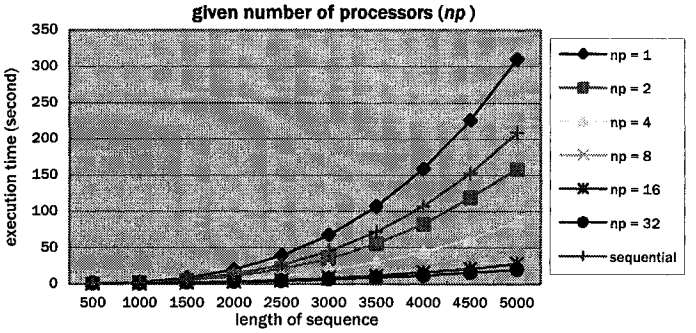

Figure 9. Execution time for various lengths of sequence

On the other hand, given specific length of sequence, the execution time of parallel algorithm tends to fall as the number of parallel processors increases, which is shown in Fig. 10. Obviously, our algorithm successfully captures the parallelism of the problem itself and unleashes the power of PC clustering systems, consequently improving the computational performance effectively. However, the parallel efficiency is found in a fairly low level partly because of the high proportion of communication for data transmission among processors.

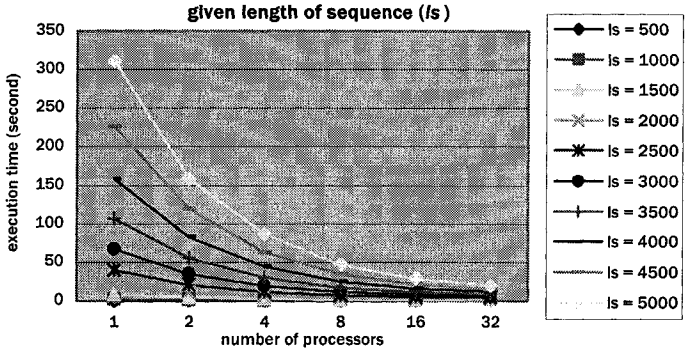

Figure 10. Execution time for various numbers of processors

5. ENCAPSULATION OF GRID SERVICE

Grid is an integrated environment for shared resources and services, which aims to establish a virtual supercomputer by organizing the geographically distributed computers all over the world via Internet.

5.1 Grid development environment

We attempt to encapsulate the core algorithm of sequence alignment into a grid service for the convenient access to end users. In doing so, we use the Globus Toolkit 3.0 (GT3), a software toolkit for building grid applications based on OGSI 1.0. Before all, a development environment is set up through the installation and configuration of JDK, Ant and GT3 Core.

5.2 Programming and deployment

We follow the proxy-stub model, a general programming model for distributed computing. It comprises two parts: the *server end* and the *client end*, which are weakly-coupled via service description.

The server end describes the service information in a file written in WSDL that includes service interface, invoking method and its binding low-level communication protocol. We rewrite the previously implemented algorithm and deploy it into a Web service container provided by GT3 itself. The client end generates the stub for service invoking according to the WSDL file from the server end. It receives the request data submitted by end users, invokes the core algorithm deployed on the server end via predefined interface and eventually returns the processed results.

In the respect of a friendly user interface, we embellish the client end using HTML and JSP technology such that users are allowed to gain entrance to the grid service of sequence alignment by means of web browsers and submit their specific tasks through online forms; the results are organized in the standard format and a concise report with statistical and analytical information is presented (shown in Fig. 11).

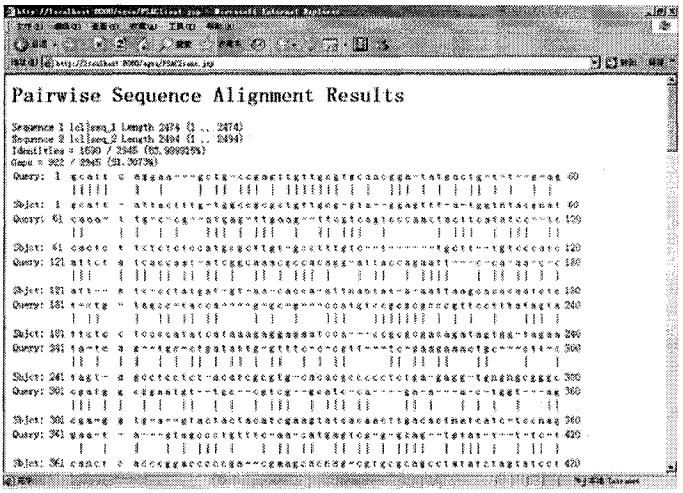

Figure 11. Obtain results of sequence alignment via grid service

6. CONCLUSION

In this work, several algorithms for sequence alignment are analyzed and a visualized program is implemented for further research. We focus on the pairwise alignment algorithm based on dynamic programming and have it parallelized for distributed memory systems based on pipeline model and wave-front method. The algorithm is implemented on a PC cluster with MPI and the experimental results indicate a remarkable enhancement of algorithm performance in comparison with the sequential algorithm. Furthermore, we encapsulate the algorithm into a grid service for practical use.

We are currently trying to discover more about the potential parallelism of algorithm especially in the backtracking procedure and also the hardware characteristics of clustering systems such as its network topology. The future work will involve transplanting the algorithm into a grid computing environment, and direct to the research of multiple alignment algorithm. Our goal is to find an efficient and scalable solution for biological sequence alignment.

7. ACKNOWLEDGMENTS

This work was supported by the Science Foundation of Shanghai Municipal Commission of Science and Technology, grant No. 00JC14052, and by "SEC E-Institute: Shanghai High Institutions Grid" project.

8. REFERENCES

1. Andreas D. Baxevanis, B. F. Francis Ouellette. "Bioinformatics: the practical guide for analysis of genomics and proteomics". Tsinghua University Press. 2000
2. Borja Sotomayor. "The Globus Toolkit 3 Programmer's Tutorial". 2004.
3. Cyntbia Gibas, Per Jambeck. "The computer technology in bioinformatics". China Dianli Press. 2002
4. C. Xavier, S.S. Iyengar. "Introduction to the parallel algorithm". China Machine Press. 2004
5. D. S. Hirschberg. "A Linear Space Algorithm for Computing Maximal Common Subsequences". Comm. ACM, vol. 18, no. 6, pp. 341-343, 1975
6. E. W. Mayers, W. Miller. "Optimal Alignments in Linear Space". Computer Applications in the Biosciences, vol. 4, no. 1, pp. 11-17, 1988
7. Joshy Joseph, Craig Fellenstein. "Grid Computing". Tsinghua University Press. 2005
8. O. Gotoh. "An Improved Algorithm for Matching Biological Sequences". J. Molecular Biology, vol. 162, pp. 705-708, 1982
9. S. B. Needleman, C. D. Wunsch. "A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins". J. Molecular Biology, vol. 48, pp. 443-453, 1970
10. T. F. Smith, M. S. Waterman. "Identification of Common Molecular Subsequences". J. Molecular Biology, vol. 147, pp. 195-197, 1981
11. Zhihui Du. "High-performance computing and parallel programming technology: MPI parallel programming". Tsinghua University Press. 2001