Loop-Structured-Computer によるアミノ酸配列ペアワイズアライメント

菊池信明, 李中民, Demelo M. Lao, Natalia V. Polulyakh, 吉岡良雄, 清水俊夫
弘前大学理工学部電子情報工学科

DP法によるアミノ酸配列ペアワイズアライメントを用いたデータベースのホモロジー検索は長時間の計算を要する問題であり、これまでいくつかの並列処理による高速化の試みがなされている。本研究では、データフロー型の並列計算機であるLoop-Structured-Computer (LSC) 上でその並列処理を行うことを試みた。ソースコードの記述は複雑で膨大となり、手動による作成は不可能に近い。そこで、与えられた２本のアミノ酸配列、ギャップコスト、一致・不一致のスコアに対するソースコードを自動的に生成するプログラムジェネレータを考案することによって、LSC上でアミノ酸配列ペアワイズアライメントの結果を正しく得ることができた。さらに、使用するPE数と計算時間の関係、速度向上率についても検討した。

Parallel Computing Technique of Pairwise Alignment by a Dataflow Loop-Structured-Computer

Nobuaki Kikuchi, Zhongmin Li, Demelo M. Lao, Natalia V. Polulyakh, Yoshio Yoshioka and Toshio Shimizu
Dep. of Electronic Information System Engineering, Faculty of Science and Technology, Hirosaki University

The pairwise sequence alignment by dynamic programming algorithm is performed in the Loop-Structured-Computer (LSC). The Processing Elements (PEs) of the LSC are connected with each other to make a loop configuration and the number of connected PEs can be varied. However, there is a drawback in using the LSC, i.e, it requires a lot of programming efforts to write the source code. To deal with this, we wrote a program that automatically generates the appropriate LSC source code with the option to vary the initial parameters: amino acid sequences lengths, gap cost, and match/mismatch weights. Pairwise alignment by the dataflow LSC is implemented successfully as confirmed by the correct alignment obtained. We also show in this paper the performance of the LSC in parallel processing involving different number of PEs.

1 Introduction

Since faster microprocessors are easily available recently, parallel computers with shared memory technology are becoming the mainstream. Yet, the problem associated in building interconnections for large-scale multiprocessors system remains unsolved.

We have proposed a parallel computer with a dataflow processing scheme in 1986, the loop-structured-computer (LSC) [1]. In the LSC, the processors are connected in a unidirectional configuration designed in the simplest form, which provides the minimum number of links and simpler hardware interface.

Pairwise sequence alignment is one of the most fundamental techniques to find similarity of biological sequences such as amino acid and nucleotide sequences. With the ongoing genome and post-genome projects, this technique is getting even more important. To keep in pace with the explosively increasing biological data, speedup of the sequence comparing techniques is required. In particular, more powerful and faster algorithms than, e.g., BLAST algorithm [2]. To satisfy this requirement, parallel processing has been considered as an alternative approach for the speedup of the sequence comparison [3-6].

2 Architecture of the LSC

The PEs of the LSC are connected in a unidirectional loop configuration as illustrated in Fig. 1 below:

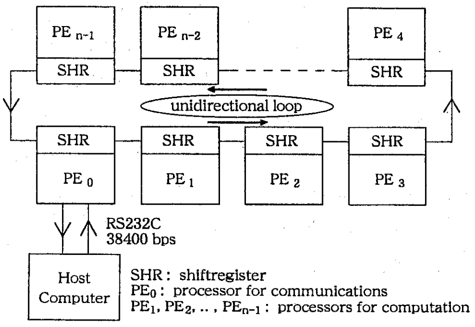

Figure 1: Architecture of the LSC

It consists of n-1 PEs for computations and one PE for communications that is connected to the host computer. Linking the shiftregisters (virtual ones) among the PEs realizes the loop connection. These virtual shiftregisters are activated through a non-maskable interrupt (NMI) routine executed at the synchronized interrupt clock of 250 Hz.

3 Pairwise Sequence Alignment by Dynamic Programming Algorithm

The pairwise sequence alignment problem can be solved rigorously by applying the dynamic programming algorithm [8, 9]. Fig. 2 illustrates a simple example of the dynamic programming algorithm for pairwise alignment.

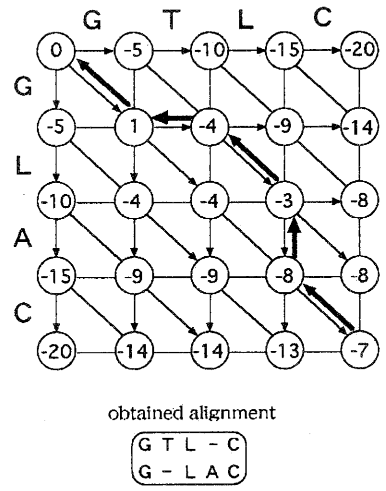

Figure 2: The path matrix for dynamic programming algorithm applied to the pairwise alignment of two amino acid sequences, "GTLC" and "GLAC".

Two sequences to be compared are placed on the horizontal and vertical axes of the matrix. The alignment is made from upper left to lower right in the matrix. The score of each node is computed from a score function. The optimal value of the score function for each node is selected from three possibilities: diagonal, horizontal and vertical paths. This procedure is represented by the score matrix, D:

$$D_{i,j} = \max(D_{i-1,j-1} + w_{i,j}, D_{i-1,j} + d, D_{i,j-1} + d) \qquad (1)$$

where $w_{i,j}$ is the weight for substituting the i-th letter of the horizontal sequence for the j-th letter of the vertical sequence (vice versa), and d is the weight for a single gap. The substitution weight is defined by the substitution matrix among 20 amino acids (e.g., Dayhoff matrix [10]). For simplicity, we defined here the weights for match, mismatch and gap as 1, -5 and -5, respectively.

4 Implementation of the Pairwise Alignment in the Dataflow LSC

4.1 Parallel Computing Technique of Pairwise Alignment by Dynamic Programming

The dynamic programming algorithm can be made more efficient by parallel processing. The standard way of computing the node, $D_{i,j}$, is in the row-wise direction, in which each node is computed from left to right for each row, and from top to bottom in the matrix. In contrast, if the order of computation is taken along the anti-diagonal direction as shown in Fig. 3 (bold arrow), the computations of $D_{i,j+1}$ and $D_{i+1,j}$ on the same anti-diagonal line can be performed simultaneously.

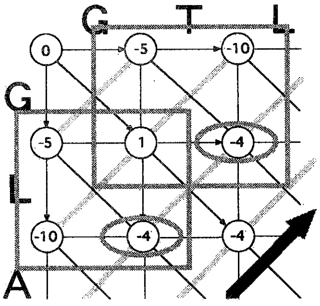

Figure 3: An illustration of a parallel processing in a dynamic programming algorithm applied to pairwise alignment.

4.2 The LSC Code for Pairwise Alignment

Programming for pairwise alignment in the LSC, fortunately, does not require writing all the routines to compute for the score of each node in the score matrix. Instead, only the code to compute for the score of one node is written and the same code is copied to compute for the scores of the other nodes, but with different variable identifier corresponding to the node subscript. Thus, to facilitate in writing the repetitive source code, we decided to write an automatic LSC source code program generator using the C language.

An example of LSC source code to compute for the score of a single node, $D_{1,1}$, is shown in Fig. 4. These codes are simply repeated except for the change in the variable's identifier.

```
                    .
                    .
                    .
ml_l=s0_0==s1_0
mml_l=true(match,ml_l)
mml_l=false(mismatch,ml_l)
gml_l=g0_0+mml_l
gil_l=g0_l+g_cost
gjl_l=g1_0+g_cost
mil_l=gml_l=>gil_l
miml1_l=true(gml_l,mil_l)
miil1_l=false(gil_l,mil_l)
mjl_l=gml_l=>gjl_l
mjml1_l=true(gml_l,mjl_l)
mjjl1_l=false(gjl_l,mjl_l)
 ijl_l=gil_l=>gjl_l
 ijil1_l=true(gil_l,ijl_l)
 ijjl1_l=false(gjl_l,ijl_l)
dfml_l=set(miml1_l,mjml1_l)
dfil_l=set(miil1_l,ijil1_l)
dfjl_l=set(mjjl1_l,ijjl1_l)
gl_l=dfml_l
gl_l=dfil_l
gl_l=dfjl_l
                    .
                    .
                    .
```

Figure 4: A sample LSC source code that computes the score of the node, $D_{1,1}$.

4.3 The Mode of Allocation of the LSC Source Code

We tested two allocation modes of the LSC source code to find out which one enhances further parallel processing in the computation. In the first mode, each line (operator) of the instruction set is allocated to each PE. On the contrary, in the second mode, the whole set of instructions for a particular node is allocated to one PE, and so on.

5 Results and Discussion

5.1 Pairwise Alignment

The pairwise alignment is successfully obtained for two sequences "GTKALAILC" and "GTLGC", by the dataflow LSC using the dynamic programming approach as shown below:

```
GTKALAILC
GT--L--GC.
```

Even if the length-dependent gap cost is used, still the correct alignment is obtained (results not shown). In the case of longer sequences, i.e., with a length of 30 amino acids, it is confirmed that the alignment can be obtained in spite of the enormous number of nodes to be processed (again results not shown). In view of the above consideration, it is possible to improve further the program generator's capability to handle longer sequences, and this is to be addressed in the future research undertaking.

We note that this is the first attempt to apply the dataflow LSC to compute for the pairwise alignment of biological sequences.

5.2 Performance of the Dataflow LSC

In here, we highlight the pairwise alignment performance of the dataflow LSC. We show the computing time and speedup ratio (T_1 / T_n, where T_1 and T_n are the computing times with one PE and n PEs, respectively) vis-a-vis the number of PEs involved in the actual computation. The number of PEs used in the computation of scores of the nodes in the score matrix, ranges from as low as eight to as high as 32 PEs.

The performance using different number of PEs in the computation of scores of the nodes, as measured in terms of the computing speed and speedup, is shown in Figures 9 and 10.

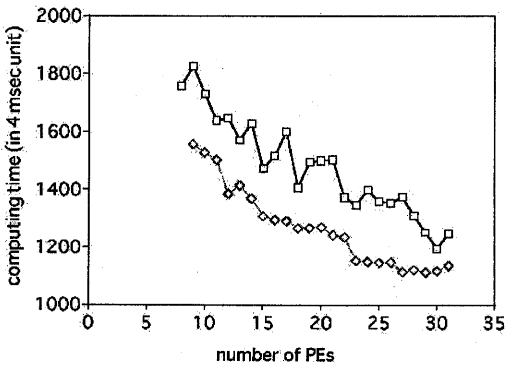

Figure 5: Speed performance of the LSC for the two allocation modes: the first mode (square) and the second mode (diamond).

Generally, as the number of PEs used in the computation increases, the computing time decreases

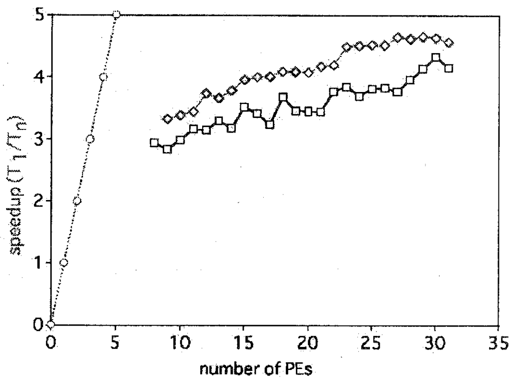

Figure 6: Speedup performance of the LSC for the two allocation modes: the first mode (square) and the second mode (diamond), as compared with the ideal case (circle).

while at the same time increases speedup in either of the two LSC allocation modes. However, applying the second mode is more efficient than the first in terms of shorter computing time and higher speedup ratio. This can be attributed to the fact that the first mode takes a little more time to consolidate the result of each PE to output the score for a particular node since several operators are spread-out to the different PEs. Moreover, it may take more than one pass in the loop to process one node. In contrast, the second mode shortens the process by performing all computations for one particular node in one PE. Thus, eliminating the time to consolidate the results for the score of one particular node by making passes in the loop. Moreover, the output for each node is obtained immediately after each execution in the PEs.

Although there was an observable improvement in processing speed while using the second allocation mode, the slopes of the two curves are nearly identical suggesting that the rate of parallel processing is barely improved. One of the reasons might be that the mode of allocation used is inefficient due to its simplicity. Perhaps, trying out novel allocation modes and improving the LSC souce code might bring forth improvement in the parallel-processing aspect. For this purpose, we plan to pursue in the next research activity the improvement of the allocation mode especially if the number of nodes is less than the number of PEs available for computation. In other words, when the LSC is underutilized, to determine which PE processes what node. Along this line, we hope to establish a new allocation mode that is efficient and capable in effecting a high rate of parallel processing.

References

[1] X. Zao, A. Narita, S. Mizuta and Y. Yoshioka, "Performance Evaluation of the LSC by Simulations of a Cascade Shower Simulation Model", In proceedings of the ISCA 12-th International Conference on Parallel and Distributed Computing Systems (1999).

[2] S. F Altschul, W. Gish, W. Miller, E. W. Myers and D. J. Lipman, "Basic local alignment search tool", J. Mol. Biol., 215, 403-410.

[3] Z. Li, Y. Yoshioka and T. Shimizu, "Pairwise Alignment on the Loop Structured Computer", Genome Informatics, 10, 342-343 (1999).

[4] X. Huang, "A Space-efficient Parallel Sequence Comparison algorithm for a Message-passing Multiprocessor", International J. of Parallel Programming, 18, 223-239 (1989).

[5] P. G.-Jamet and D. Lavenier, "SAMBA: hardware accelerator for biological sequence comparison", CABIOS, 13, 609-615 (1997).

[6] O. Trelles, M. A. Andrade, A. Valencia, E. L. Zapata and J. M. Carazo, "Computational space reduction and parallelization of a new clustering approach for large groups of sequences", BIOINFORMATICS, 14, 439-451 (1998).

[7] J. B. Dennis and D. P. Misunas, "A Preliminary Architecture for a Basic Data-Flow Processor", Laboratory for Computer Science, MIT, CSG Memo 102, 27 (1970).

[8] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins.", J. Mol. Biol., 48, 443-453. (1970).

[9] M. Kanehisa, "Post-genome Informatics", Oxford Univ. Press (2000).

[10] M. O. Dayhoff, R. M. Swchwartz and B. C. Orcutt, "A model of evolutionary change in proteins", in Atlas of protein sequence and structure, 5 (3), 345-352 (1978).