PARALLEL MULTIPLE SEQUENCE ALIGNMENT USING SPECULATIVE COMPUTATION

Tieng K. Yap[1], Peter J. Munson[1], Ophir Frieder[2], and Robert L. Martino[1]

[1]Division of Computer Research and Technology, National Institutes of Health
12 South Dr. • Bldg 12A • Rm 2033 • Bethestha, MD 20892-5624
{yap | munson | martino}@alw.nih.gov
[2]Department of Computer Science, George Mason University
Fairfax, VA 22030-4444
ophir@cs.gmu.edu

Appeared in the

Proceedings of the
1995 International Conference on Parallel Processing

August, 1995

PARALLEL MULTIPLE SEQUENCE ALIGNMENT USING SPECULATIVE COMPUTATION

Tieng K. Yap[1], Peter J. Munson[1], Ophir Frieder[2], and Robert L. Martino[1]

[1]Division of Computer Research and Technology, National Institutes of Health
12 South Dr. • Bldg 12A • Rm 2033 • Bethestha, MD 20892-5624
{yap | munson | martino}@alw.nih.gov
[2]Department of Computer Science, George Mason University
Fairfax, VA 22030-4444
ophir@cs.gmu.edu

Abstract -- *Many different methods have been presented for aligning multiple biological sequences. These methods can be classified into three categories: rigorous, tree-based, and iterative. The rigorous method, which always generates the optimal alignment, requires memory space and computation time proportional to the product of the sequence lengths. Even for a modest number of sequences, this method becomes impractical. As a result, the other two methods were introduced. The iterative methods were shown to generate better alignments than the tree-based methods. However, these methods require as much as 100 times longer computation time. A number of days may be required to align a large number of sequences (e.g., over 100 sequences) sequentially. We present a parallel speculative computational method which reduces the computation time of the iterative methods from days to minutes. To evaluate our parallel method, we implemented a speculative computation version of the iterative improvement method of Berger and Munson on an Intel iPSC/860 parallel computer. The empirical results demonstrate that our parallel method obtained a significant speed up in comparison to the sequential method.*

INTRODUCTION

Pairwise sequence alignment is an important tool for locating similarity patterns between two biological (DNA and protein) sequences. This analytical tool has been used successfully to predict the function, structure, and evolution of biological sequences. The first biological sequence alignment algorithm, referred to as dynamic programming, was introduced by Needleman and Wunsch [[21]]. This algorithm was later improved by a number of researchers through an improved formulation [[24]], a reduced time complexity [[8]], and a reduced space complexity [[18]]. As more sequences were generated by the biomedical community, researchers began to use multiple sequence alignment to obtain a better understanding of biological sequences. Although the rigorous dynamic programming algorithm can be extended to align more than two sequences, it is impractical to extend this algorithm to the alignment of more than three sequences [[19], [20]] since memory space and computation time is proportional to the product of the sequence lengths. By restricting the number of possible alignments, the algorithm can be extended to align six to eight sequences [[4], [16]].

To practically align a large number of sequences, many researchers use the tree-based methods which generate a multiple sequence alignment by combining a number of pairwise alignments in a particular order. For a binary tree where there is only one branch at each level, the order is linear [[1], [17], [25]]. These linear ordering strategies start with the most similar sequence pair and continue to add sequences to the alignment in order of decreasing similarity. The linear ordering strategies produce a good multiple alignment if all the sequences belong to a single homologous family. However, as pointed out by Taylor [[26]], these strategies can produce a poor alignment if the sequences belong to two or more distinct subfamilies.

To improve the tree-based methods, researchers introduced sophisticated ordering strategies [[2], [5], [7], [9], [10], [25]]. These strategies apply various clustering techniques to order groups of related sequences in a hierarchical tree. Then, the final multiple alignment is obtained by combining clusters of

[2]This author was partially supported by the National Science Foundation under contract number IRI-9357785, by the Virginia Center for Innovative Technology under contract number INF-94-002, and by XPAND Corporation.

sequences to the most similar cluster in decreasing order of relatedness. Using tree-based methods, the order used to align and combine the sequences has a great effect on the final multiple sequence alignment. Consequently, a great deal of effort has been spent on designing new ordering strategies that would generate better alignments.

A few researchers [[3], [12], [14], [15]] have taken the opposite approach by not ordering the sequences in a systematic way. Instead, they applied randomized techniques with optimization functions to iteratively improve the multiple sequence alignment. These iterative methods produce better alignments than the tree-based methods and can be used to improve alignments that were generated from a tree-based method. However, they require a much longer computation time. In this paper, we use the Berger-Munson algorithm [[3]] to illustrate that the computation times of these iterative methods can be reduced significantly by using a parallel speculative computation technique.

SEQUENCE ALIGNMENT ALGORITHMS

Needleman and Wunsch [[21]] were the first to introduce a heuristic algorithm for aligning two biological sequences. Smith and Waterman [[24]] then formulated a more rigorous dynamic programming representation of this algorithm. Gotoh [[8]] followed by improving the time complexity of the algorithm. Let two sequences be $A=a_1a_2a_3...a_M$, and $B=b_1b_2b_3..b_N$ and let $sub(a_i,a_j)$ be a given similarity score for substituting residue a_i by a_j. The penalty for introducing a gap into a sequence is defined by the penalty function $w_k=-uk-v$, $(u,v\geq 0)$ where k is the gap length. The Gotoh algorithm is given as follows:

$$S_{i,j} = MAX\begin{cases} P_{i,j} \\ S_{i-1,j-1} + sub(a_i, b_j) \\ Q_{i,j} \end{cases}$$

$$P_{i,j} = MAX\begin{cases} S_{i-1,j} + w_1 \\ P_{i-1,j} + u \end{cases}$$

$$Q_{i,j} = MAX\begin{cases} S_{i,j-1} + w_1 \\ Q_{i,j-1} + u \end{cases}$$

$S_{i,j}$ is the cumulative alignment score between two sequences A and B up to the i*th* and j*th* positions. The initial conditions are defined as follows: $S_{i,0}=P_{i,0}=Q_{i,0}=0$ and $S_{0,j},P_{0,j}=Q_{0,j}=0$, for $0\leq i\leq M$ and $0\leq j\leq N$. As can be seen, the similarity scores *sub*(a,b) between all possible pairs of residues must first be defined before $S_{i,j}$ can be calculated. For our application, we use the PAM250 [[6]] substitution scoring matrix to define the similarity score between two residues.

To obtain the multiple sequence alignment, many algorithms (including tree-based and iterative) align two groups of sequences against each other a number of times. To align two groups, X and Y, of sequences, the algorithm of two-sequence alignment can be extended as follows:

$$S_{i,j} = MAX\begin{cases} P_{i,j} \\ S_{i-1,j-1} + sub'(X_i, Y_j) \\ Q_{i,j} \end{cases}$$

$$sub'(X_i, Y_j) = \sum_{k=1}^{K}\sum_{l=1}^{L} sub(X_{k,i}, Y_{l,j}) + \sum_{k=1}^{K}\sum_{m=k+1}^{K} sub(X_{k,i}, X_{m,i}) + \sum_{l=1}^{L}\sum_{m=l+1}^{L} sub(Y_{l,j}, Y_{m,j})$$

K is the number of sequences in the X group and L in the Y group. $P_{i,j}$, $Q_{i,j}$, and the initial conditions remain the same. If the sequences have already been aligned, the alignment score can be calculated using the following formula.

$$Score(A_N) = \sum_{i=1}^{N}\sum_{j=i+1}^{N}\sum_{k=1}^{L} sub(A_{i,k}, A_{j,k})$$

where N is the number of sequences and L is the number of aligned positions.

BERGER-MUNSON ITERATIVE ALGORITHM

The Berger-Munson iterative improvement algorithm has been successfully used to perform multiple sequence alignment. Figure 1 shows the core part of this algorithm. The C language implementation contains approximately 1900 statements. To implement a parallel version of this algorithm, we separated the computational process into three steps. In step 1, the *n* input sequences are first randomly partitioned into two groups. Then, the alignment score between these two groups of sequences is calculated. In this step, the new gap positions are also saved for performing the alignment in step 3. In step 2, a decision flag is set to A (accepted) if the new resulting alignment is accepted; otherwise, it is set to R (rejected). A new alignment is accepted if the current score is higher than the current best score. If the decision flag in step 2 is set to A, the gap positions

determined in step 1 are used to modify the current alignment in step 3 and the best score is updated. Then, the modified or unmodified alignment is used as the input for the next iteration. This iterative improvement algorithm continues until the stop criterion is met. We define the stop criterion as follows. After *q* consecutive iterations of rejections, the process is stopped where *q* is the number of all possible partitions.

```
best_score=initial_score();
While (stop criteria is not met){
    1 current_score = calculate(seq, gap_positions);
    2 flag = decide(current_score, best_score);
    3 seq = modify(seq, flag, gap_positions);
}
```

Figure 1. Berger-Munson Sequential Algorithm.

The Berger-Munson algorithm is highly sequential due to a loop-carried dependence between iterations. Iteration *i* depends on iteration *(i*-1*)* since step 3 may modify the alignment during the *(i*-1*)th* iteration and the modified alignment must be used by the *i*th iteration. In addition, the three steps within each iteration are also dependent on each other. Step 1 uses *seq* which may be modified by step 3. Step 2 uses *current_score* which produces by step 1. Step 3 uses *flag* variable which is set in step 2 and gap positions which are generated in step 1. These dependencies make it difficult to implement a parallel version of this algorithm while preserving the behavior of the original sequential version.

REVIEW OF A PRIOR BERGER-MUNSON PARALLELIZATION EFFORT

Ishikawa *et al.* [[11]] previously implemented a parallel version of the Berger-Munson algorithm on a parallel inference machine (PIM) using a parallel logic programming language (KL1). This parallel approach can be described as follows. All $(2^{n-1} - 1)$ possible partitions or $n + \frac{n(n-1)}{2}$ restricted partitions, which are defined later in the Alignment Search Space section, are evaluated simultaneously in parallel. The resultant alignment which has the best score is selected as the input for the next iteration. One processor is used as the manager and the remaining processors as workers. Initially, the manager distributes each possible partition to a worker. When a worker finishes its calculation, it sends its alignment to the manager. Based on all the alignments collected from the workers, the manager selects the best alignment to be used as the input for the next iteration.

The approach taken by Ishikawa *et al.* [[11]] has a few drawbacks. First, it becomes impractical for a large number of sequences. For example, approximately 10^{90} processors are needed to align 300 sequences if the unrestricted search space is used or 45,150 processors for the restricted space. Their implementation can be modified so that a large number of sequences can be aligned by dividing the number of partitions among the available processors as evenly as possible. However, it is still too costly to evaluate all partitions at each iteration. In the sequential version, only one random partition is evaluated at each iteration. Second, the parallel version is no longer a randomized process and its resultant alignment is not guaranteed to be as good as the one that is obtained from the original sequential version. That is, the quality of the derived alignment is unpredictable. Therefore, it is difficult to evaluate its performance. Third, the communication cost of the Ishikawa version can be reduced significantly.

In their approach, the sequences were sent back and forth between the manager and workers twice per parallel iteration. The manager sends the input sequences with the partition information to all the workers at the beginning of each iteration and all the workers send their alignments to the manager when they are done. As a result, the communication cost per iteration is approximately 2*pnm*, where *p* is the number of processors, *n* is the number of sequences, and *m* is the length of the longest aligned sequence (original residues plus gaps). In our approach, we do not send sequences between processors. Only the gap and partition information are sent. As a result, our communication cost is approximately 2*m* since only one array of length *m* is used to hold the gap positions of each group. For large *p* and *n*, our communication cost is negligible in comparison to their approach.

PARALLEL SPECULATIVE BERGER-MUNSON ALGORITHM

Speculative computation [[22], [23], [27]] has been applied efficiently to parallelize sequential algorithms like simulated annealing, an algorithm similar to the Berger-Munson algorithm. By applying speculative computation to the parallelization of the Berger-Munson algorithm, we were able to achieve a higher speedup and a more scalable implementation than the prior effort mentioned above. In addition, our parallel alignment is guaranteed to be the same as the sequential one. The basic concept of speculative

Sequential Iteration number	1 2 3 4 5 6 7 . . .
Decision sequence	A A A A A R A A A R R A R R R A R R R A R R R A R R R R ...

Figure 2. A Possible Sequential Decision Sequence.

computation is to speculate the future solutions based on the current input parameters. Therefore, we can speculate (*p*-1*)* future solutions if we have *p* processors. In this application, we can speculate the alignments for the next (*p*-1*)* iterations based on the current alignment.

In the original Berger-Munson algorithm, the final alignment is obtained by performing a sequence of alignments between two groups of sequences. Each iteration is accepted (A) if its alignment score is higher than the current best score. Otherwise, it is rejected (R). An example of a corresponding sequence of decisions is shown in Figure 2. Initially, every new alignment is accepted (e.g., iteration numbers 1-5). However, fewer and fewer are accepted as the alignment progresses. We stated earlier that the *i*th iteration may depend on the *(i*-1*)*th iteration. To be exact, the *i*th iteration depends on *(i*-1*)*th iteration only if the *(i*-1*)*th iteration has accepted a new alignment; otherwise, it only depends on the last accepted iteration.

Our parallel speculative computation approach is based on the recognition of the fact that a consecutive sequence of rejected iterations are not dependent on each other and can be done in parallel. Therefore, we can speculate that the (*p*-1*)* previous iterations will be rejected so that they can be done in parallel. If the speculations are correct, the computation time could be reduced by a factor of *p*.

In the decision sequence of Figure 2, the first 28 sequential iterations can be reduced to 13 parallel steps if 4 processors are used. The parallel computation steps are shown in Figure 3. Three iterations (*p*-1) are speculated at each parallel step where P_1 speculates that P_0 will reject its new alignment, P_2 speculates that P_0 and P_1 will reject their new alignments, and P_3 speculates that P_0 to P_2 will reject their new alginments. P_0 does not speculate. The numbers in the boxes of each row represent the speculated sequential iteration numbers for the processor in that row at each parallel step. The iteration numbers that are speculated correctly, which also correspond to the sequential iteration number, are shown in bold. After each parallel step, the alignment of the last iteration that was speculated correctly is used as the input for the next step as shown by the lines leading from one parallel step to the next.

As the above illustration shows, we parallelized the Berger-Munson algorithm while preserving its sequential algorithmic behavior. The parallel alignment is guaranteed to be the same as the sequential one. For a large number of sequences, this algorithm can benefit significantly from parallel computation. Our parallel algorithm, which is implemented on every processor, is summarized in Figure 4 as C pseudocode with minor details omitted to improve clarity.

The variable *gi* is the global or sequential iteration number; *bgi* is the iteration number when the best score was obtained; *q* is the number of all possible partitions; *partn* is a selected partition number for each individual processor; *p* is the number of processors; *pid* is the processor id ranging from 0 to (*p*-1) and *ap* is the id of the processor that has accepted the best alignment.

To reduce the I/O time, only processor 0 reads the input sequences and then broadcasts them to the other processors since inter-processor data transfer is much faster than the I/O data transfer. Initially, every new alignment is usually accepted. As a result, we do not start to speculate until we encounter a rejection, (see lines 4 to 11). Every processor evaluates the same partition by initializing the same random seed. This

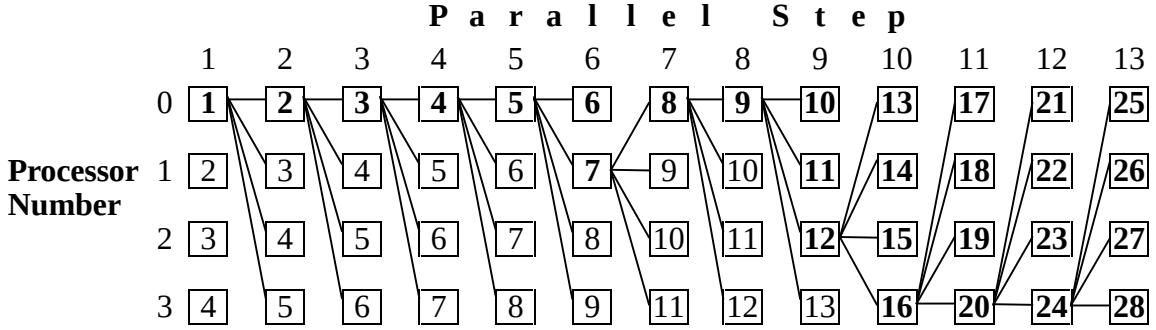

Figure 3. An Illustration of the Parallel Speculative Computation Process.

strategy avoids the communication cost associated with the parallel speculative computation. The iteration number is used as the random seed so that we can easily backtrack our steps when we make an incorrect speculation. This technique is also used to guarantee that the sequence of pairwise alignments is the same for both the parallel and sequential implementations.

```
1   processor 0 reads input sequences and broadcasts them to
    other processors.
2   gi = 0;
3   best_score = initial_score();
4   Do {
5      seed(gi);  /*all processors set the same iteration seed gi*/
6      partn = select_partition();
7      current_score = calculate(seq, partn, gap_positions);
8      flag = decide(current_score, best_score);
9      seq = modify(seq, partn, flag, gap_positions);
10     gi = gi + 1;
11  }While (flag == A);
12  clear_partitions();
13  While((gi - bgi) < q){
14  for(i = 0; i < p; i++){
15     seed(gi + i);
16     itemp = select_partition();
17     set_partition(itemp);
18     if (i == pid) partn = itemp;
19  }
20  current_score = calculate(seq, partn, gap_positions);
21  flag = decide(current_score, best_score);
22  global_operation(ap, flag, best_score, partn, gap_positions);
23  seq = modify(seq, partn, flag, gap_positions);
24  if(flag == A){
25        gi = gi + ap + 1;
26        clear_partitions();
27  }
28  else gi = gi + p;
```

Figure 4. Parallel Speculative Berger-Munson Algorithm.

After a rejection is encountered, we start to speculate and continue until q (number of all possible partitions) rejections have been encountered. A random partition is selected only if it has not already been selected since the last accepted partition. That is, no partition is selected more than once by any processor or by different processors simultaneously. To ensure that no partition is selected more than once, each processor must know two pieces of information: the partitions that have already been selected and the partitions that are currently being selected by other processors. To avoid the costly inter-processor communication, these two pieces of information are obtained as follows. Each processor manages an array of q bits which correspond to the q possible partitions. Initially, these bits are cleared by a function in line 12. Then, the ith bit is set when the ith partition is selected. Therefore, each processor knows that a particular partition has already been selected if its corresponding bit is set. When this situation occurs, it simply selects another random partition. All q bits are cleared every time a new partition is accepted. To determine the partitions that are being selected by other processors, each processor generates p random selectable partitions instead of just one and then selects the (pid)th one as show in lines 14-19. The remaining partitions are being selected by the other processors. All p bits that are corresponding to the p selectable partitions are set.

The global operation (line 22) is performed after each processor makes its decision. The accepted alignment with the smallest iteration number is selected as the input for the next iteration since the alignments with higher iteration numbers are invalid. That is, they were based on incorrect speculations. When a new partition is accepted, the contents of variables *(ap, flag, best_score, partn, gap_posititons)* are copied from the accepted processor to the other processors.

In lines 24-28, we determine the number of sequential iterations which were correctly speculated for skipping. If there is a global accepted partition (iteration) among the p partitions evaluated, only the iterations smaller than or equal to the accepted iteration are skipped (line 25). Otherwise, all p iterations are skipped (line 28).

ALIGNMENT SEARCH SPACE

We adapted the restricted search space as presented by Ishikawa *et al.* [[11]] who observed that the number of sequences in the divided groups had a great effect on the final alignment. They observed that if only one or two sequences were allowed in the first group, a better alignment was obtained. Our experiments yield the same observation. If only one or two sequences are allowed in one of the two groups, the number of possible partitions is reduced to $n + \frac{n(n-1)}{2}$ from a total of $2^{n-1} - 1$. We represent a restricted partition by a single random number. This number is then used to generate a sequence of 2 or 3 unique numbers. The first number is the number of sequences in the first group and the following one or two numbers representing the sequence numbers in this group. The remaining sequences are placed into the second group.

RESULTS

To evaluate the performance of our approach, we have used it to improve the alignments generated manually by experts, Kabat *et al.* [[13]], and automatically by a popular program, CLUSTALV [[9],

[10]], which uses a tree-based method. Three different groups of immunoglobulin sequences with varying lengths and numbers of sequences were selected from the Kabat Database (Beta Release 5.0) which is maintained by Kabat *et al* [[13]]. Their statistical summaries are shown in Table 1. The average sequence length is about the same for all three groups. However, the number of sequences in the third group is about twice the second one which is about twice the first one. MKL5 is the largest group in this database. CLLC is the chicken immunoglobulin lambda light chains V-region group. HHC3 is the human immunoglobulin heavy chains subgroup III V-region group. MKL5 is the mouse immunoglobulin kappa light chains V V-region group. The initial score is the score before any alignment is performed.

Table 1. Statistical Summaries of the Three Groups of Test Sequences.

Group Name	Number of Sequences	Average Length	Initial Score (10^3)
CLLC	93	62	1,575
HHC3	185	65	6,651
MKL5	324	83	31,394

The scores of the alignments manually generated by experts, Kabat *et al.* [[13]] and their improved scores performed by the sequential Berger-Munson program, MUSEQAL, are shown in Table 2. The number of iterations and the sequential computation times taken by MUSEQAL are also shown in this table. These sequential computation times were obtained from executing a sequential program on a single processor. Similarly, we used MUSEQAL to improve the alignments generated by CLUSTALV. The corresponding information is presented in Table 3. Comparing Table 2 and Table 3, we can see that MUSEQAL improved the alignments generated by both Kabat *et al.* and CLUSTALV significantly. The sequential computation times in these tables are used to calculate the speedup factors of the parallel speculative Berger-Munson algorithm in the next table.

Table 4 shows the speedup factors for the three groups of sequences on varying numbers of processors. The speedup factor is defined as the ratio of the total run time of the sequential version of the program to the total run time of the parallel version. Table 4 demonstrates that significant speedups were obtained for all three groups of sequences. From this table, we can make three observations. First, we obtained the best speedup factors with the largest group, MKL5. Second, we obtained better efficiencies by using a smaller number of processors where efficiency is defined as the ratio of the speedup factor to the number of processors. Third, we achieved higher speedup factors when improving the Kabat alignments compared to CLUSTALV alignment improvement. The results of the first two observations are as expected. The first observation was due to a larger number of partitions (search space) and the second to less communication and lower rates of incorrect speculations. For a larger number of processors, we had to speculate further into the future which resulted in a higher error rate. The third observation is due to the fact that the Kabat alignments were already better than the CLUSTALV alignments. As a result, there were more rejections in improving the Kabat alignments than the CLUSTALV alignments.

Table 2. Kabat and MUSEQAL Alignment Score Comparison

Group Name	Kabat Score (10^3)	MUSEQAL Score (10^3)	Number of Iterations	Sequential Run Time (10^3 s)
CLLC	1,827	1,957	16,193	55
HHC3	7,505	7,681	56,232	567
MKL5	38,569	38,766	112,374	2,535

Table 3. CLUSTALV and MUSEQAL Alignment Score Comparison.

Group Name	CLUSTALV Score (10^3)	MUSEQAL Score (10^3)	Number of Iterations	Sequential Run Time (10^3 s)
CLLC	1,809	1,957	12,716	36
HHC3	7,366	7,655	58,285	564
MKL5	37,778	38,749	112,390	2,611

Table 4. Parallel Speculative Berger-Munson Algorithm Speedup Factors.

Number of Processors	Speedup w.r.t Kabat Alignment			Speedup w.r.t CLUSTAL Alignment		
	CLLC	**HHC3**	**MKL5**	**CLLC**	**HHC3**	**MKL5**
1	1.0	1.0	1.0	1.0	1.0	1.0
2	1.8	1.9	1.9	1.8	1.8	1.9
4	3.4	3.4	3.8	3.3	3.3	3.7
8	6.4	6.2	7.3	6.1	5.9	7.1
16	11.6	11.4	14.1	10.7	10.7	13.6
32	19.5	20.8	28.3	17.0	19.4	26.3
64	29.5	38.1	53.1	23.8	35.1	50.7

DISCUSSION

The Berger-Munson iterative method is a good tool for improving the alignments generated by other methods. As an improvement tool, it can never generate a worse alignment than other methods. Its computation time is significantly reduced by using a parallel speculative computation technique. We also

think that the computation of other iterative methods [[3], [12], [14], [15]] can also be reduced by using a parallel speculative computation approach.

It is difficult to accurately compare our speedup factors with that obtained by Ishikawa *et al.* [[11]] since their parallel algorithm does not always generate the same alignment as the sequential one. To evaluate their parallel algorithm, they aligned 7 sequences which has 63 possible partitions that were assigned to 63 processors. Their parallel implementation stops after no improvement was obtained. Their sequential (single processor) implementation stops after 32 iterations of no improvements, about one half of the number of partitions. For the sequential execution, they used different random seeds to generate different alignments. On average, they obtained a speedup factor of ten. We obtained a speedup range of 23 to 53 with our speculative method.

In spite of the above difficulty, our results clearly showed that our speedup factors are about three to five times higher than those obtained by Ishikawa *et al*. In addition, we were able to achieve higher speedup factors without changing the algorithmic behavior of the original sequential algorithm.

REFERENCES

[1] G.J. Barton, and M.J.E. Sternberg, "A Strategy For The Rapid Multiple Alignment Of Protein Sequences," *J. Mol. Bio.*, (1987), pp. 327-337.

[2] G.J. Barton, "Protein Multiple Sequence Alignment And Flexible Pattern Matching," *Methods Enzymol*., (1990), pp. 403-427.

[3] M.P. Berger, and P.J. Munson, "A Novel Randomized Iterative Strategy For Aligning Multiple Protein Sequences," *Comput. Appl. Biosci*., (1991), pp. 479-484.

[4] H. Carillo, and D. Lipman, "The Multiple Sequence Alignment Problem In Biology," *SIAM J. Appl. Math*., (1988), pp. 197-209.

[5] F. Corpet, "Multiple Sequence Alignment With Hierarchical Clustering," *Nucleic Acids Research*, (1988), pp. 10881-10891.

[6] M.O. Dayhoff, R.M. Schwartz, and B.O. Orcutt, "A Model Of Evolutionary Change In Proteins," In Dayhoff (ed) , *Atlas of Protein Sequence and Structure Vol. 5, Suppl. 3, Nat. Biomed. Res. Found*., Washington, D.C., (1978), pp.345-352.

[7] D.F. Feng, and R.F. Doolittle, "Progressive Alignment And Phylogenetic Tree Construction Of Protein Sequences," *Methods Enzymol*., (1990), pp. 375-387.

[8] O. Gotoh, "An Improved Algorithm For Matching Biological Sequences," *J. Mol. Biol*., (1982), pp. 705-708.

[9] D.G. Higgins, and P.M. Sharp, "CLUSTAL: A Package For Performing Multiple Sequence Alignment On A Microcomputer," *Gene*, (1988), pp. 237-244.

[10] D.G. Higgins, and P.M. Sharp, "Fast And Sensitive Multiple Sequence Alignments On A Microcomputer," *Comput. Appl. Biosci*, (1989), pp. 151-153.

[11] M. Ishikawa, M. Hoshida, M. Hirosawa, T. Toya, K. Onizuka, and K. Nitta, "Protein Sequence Analysis By Parallel Inference Machine," *Proceedings of the international conference on fifth generation computer systems*, (June, 1992), pp. 57-62 and 294-299.

[12] M. Ishikawa, T. Toya, M. Hoshida, K. Nitta, A. Ogiwara, and M. Kanehisa, "Multiple Sequence Alignment By Parallel Simulated Annealing," *Comput. Appl. Biosci*., (1993), pp. 267-273.

[13] E.A. Kabat, T.T. Wu, H.M. Perry, K.S. Gottesman, and C. Foeller, "Sequence Of Proteins Of Immunological Interest," *U.S. Dept. of Health and Human Services, Public Health Service, National Institutes of Health*, NIH Publication No. 91-3242, (1991).

[14] J. Kim, S. Pramanik, and M.J. Chung, "Multiple Sequence Alignment Using Simulated Annealing," *Comput. Appl. Biosci*., (1994), pp. 419-426.

[15] C.E Lawrence, S.F. Altschul, M.S. Boguski, J.S. Liu, A.F. Neuwald, J.C. Wootton, "Detecting Subtle Sequence Signals: A Gibbs Sampling Strategy For Multiple Alignment," *Science*, (1993), pp. 208-214.

[16] D.J. Lipman, S.F. Altschul, and J.D. Kececioglu, "A Tool For Multiple Sequence Alignment," *Proc. Natl. Acad. Sci.* USA, (1989), pp. 4412-4415.

[17] H.M. Martinez, "A Flex Multiple Sequence

Alignment Program," *Nucleic Acids Research*, (1988), pp. 1683-1691.

[18] E. Myers, and W. Miller, "Optimal Alignments in Linear Space," *Comput. Appl. Biosci*., (1988), pp. 11-17.

[19] M. Murata, J.S. Richardson, and J.L. Sussman, "Simultaneous Comparison Of Three Protein Sequences," *Proc. Natl. Acad. Sci. USA*, (1985), pp. 3073-3077.

[20] M. Murata, "Three-way Needleman-Wunsch algorithm," *Methods Enzymol*., (1990), pp. 365-375.

[21] S.B. Needleman, and C.D. Wunsch, "A General Method Applicable To The Search For Similarities In The Amino Acid Sequences Of Two Proteins," *J. Mol. Biol*. , (1970), pp. 443-453.

[22] A. Sohn, Z. Wu, and X. Jin, "Parallel Simulated Annealing By Generalized Speculative Computation," *Proceedings of the Fifth IEEE Symposium on Parallel and Distributed Processing*, Dallas, Texas, (December 1993).

[23] A. Sohn, "Parallel Speculative Computation Of Simulated Annealing," *Proceedings of the International Conference on Parallel Processing*, August, (1994), pp. III8-11.

[24] T.F. Smith, and M.S. Waterman, "Identification Of Common Molecular Subsequence," *J. Mol. Biol*., (1981), pp. 195-197.

[25] W.R. Taylor, "Multiple Sequence Alignment By Pairwise Algorithm," *Comput. Appl. Biosci*., (1987), pp. 81-87.

[26] W.R. Taylor, "A Flexible Method To Align Large Numbers Of Biological Sequences," *J. Mol. Evol*., (1988), pp. 161-169.

[27] E.E. Witte, R.D. Chamberlain, and M.A. Flanklin, "Parallel Simulated Annealing Using Speculative Computation," *IEEE Transactions on Parallel and Distributed Systems*, (1991), pp. 483-494.