Parallel Computation in Biological Sequence Analysis

Tieng K. Yap, Ophir Frieder, *Senior Member*, *IEEE*, and Robert L. Martino, *Member*, *IEEE*

Abstract—A massive volume of biological sequence data is available in over 36 different databases worldwide, including the sequence data generated by the Human Genome project. These databases, which also contain biological and bibliographical information, are growing at an exponential rate. Consequently, the computational demands needed to explore and analyze the data contained in these databases is quickly becoming a great concern. To meet these demands, we must use high performance computing systems, such as parallel computers and distributed networks of workstations. We present two parallel computational methods for analyzing these biological sequences. The first method is used to retrieve sequences that are homologous to a query sequence. The biological information associated with the homologous sequences found in the database may provide important clues to the structure and function of the query sequence. The second method, which helps in the prediction of the function, structure, and evolutionary history of biological sequences, is used to align a number of homologous sequences with each other. These two parallel computational methods were implemented and evaluated on an Intel iPSC/860 parallel computer. The resulting performance demonstrates that parallel computational methods can significantly reduce the computational time needed to analyze the sequences contained in large databases.

Index Terms—Sequence, comparison, alignment, search, retrieval, database, algorithm, parallel, speculative, computation.

1 Introduction

The field of molecular biology has created many specialized databases which contain diverse information, such as annotated biological sequences, three-dimensional molecular structures, and both genetic and physical maps. Keen et al. [31] have compiled a list of molecular biology databases (LiMB database) which presently contains 189 entries and is continuing to grow. The LiMB database listed 36 sequence databases, including the internationally well-known DNA sequence databases GenBank [5], EMBL [17], and DDBJ [15], and the protein sequence databases PIR [19], SWISS-PROT [1], and PDB [7].

Given the great deal of knowledge that can be derived from analyzing biological sequences, we developed parallel methods to reduce the time required to perform two computationally intensive analyses: homologous sequence searching and multiple sequence alignment. Our parallel searching method reduces the retrieval time by nearly a factor of N, where N is the number of processors. As shown in this paper, a retrieval on the GenBank that took approximately 168 hours using a single processor was reduced to only 2.6 hours using 64 processors. Our parallel alignment method also reduces the computation time significantly. Again, as shown in this paper, to align 324 protein sequences, it took about 29 days on a single processor, but only about 13 hours on a 64 processor system. These parallel computational methods are also highly scalable. That is, they can be implemented on either a small or a large number of processors. They can also be implemented on either a parallel computer or a network of workstations.

Retrieving homologous sequences from existing databases is important to the biomedical research community. When scientists discover new sequences, they are eager to search these databases for sequences that are similar or related to these discoveries. The biological information associated with the similar sequences found in the databases may provide important clues for determining the structure and function of the newly discovered ones. The detail similarity patterns of the retrieved sequences can be seen in a multiple sequence alignment, which is obtained by stacking up sequences on top of each other. A minimal number of gaps are introduced in the sequences so that the number of matches is maximized according to a given optimization function. This analysis has been used successfully to predict the function, structure, and evolutionary history of biological sequences. Multiple sequence alignment is also used in AIDS and cancer studies, where it has, for example, been used to develop AIDS vaccine components [48].

The remainder of this paper is organized as follows: First, we present algorithms used to perform biological sequence analysis. Then, we present the parallel computational methods for retrieving homologous sequences from large biological databases. In the following section, we present heuristic methods for aligning multiple sequences. We conclude by summarizing the results of our methods in the last section.

- *T.K. Yap and R.L. Martino are with the National Institutes of Health, Division of Computer Research and Technology, Bldg. 12A Room 2033, 12 South Dr. MSC 5624, Bethesda, MD 20892. E-mail: yap@helix.nih.gov, martino@alw.nih.gov.*
- *O. Frieder is with the Division of Electrical and Computer Science and Engineering, Florida Institute of Technology, 150 W. University Blvd., Melbourne, FL 32901-6988. E-mail: ophir@ee.fit.edu.*

Manuscript received 10 Apr. 1996; revised 12 Sept. 1997. For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 100177.

2 Biological Sequence Analysis Algorithms

To compare two DNA, RNA, or protein sequences, the similarities between the individual residues (nucleic or amino acids) in these sequences must first be defined. A substitution scoring matrix that defines the similarity score between all possible pairs of residues is required. Two different types of matrices are used: one for comparing the nucleic acid sequences and the other for the amino acid sequences. These matrices contain the score for substituting or exchanging one residue by another. The substitution matrices for amino acids were described in detail by Dayhoff et al. [12]. These matrices were constructed based on a combined effects of several biological factors. For the nucleic acids, the identity matrix can be used [51]. That is, a score of one unit is given for substituting one nucleic acid by another identical one. Otherwise, a score of zero is given.

A useful algorithm that gives a measure of similarity between two sequences must consider change (substitution), deletion, and insertion of residues since biological sequences are known to mutate as they evolved from one generation to the next. To define the terms substitution, deletion, and insertion more precisely, consider an example. Let sequence A = CSTPGND and sequence B = CSDTND. One possible comparison consists of the following substitutions, deletions, and insertions.

```
CS-TPGND
CSDTN-D
```

The process of comparing two sequences by writing one sequence above the other is referred to as aligning the two sequences. In the above alignment, we say that $b_3 = D$ is inserted into the first sequence or it is deleted from the second one, depending on the point of view. $a_4 = P$ is substituted by $b_5 = N$ or b_5 by a_4, again depending on the point of view. Consecutive dashes in the sequences represent a gap. Therefore, there is a gap of length one between a_2 and a_3 in the first sequence and a gap of length two between b_5 and b_6 in the second sequence.

The first algorithm for determining the optimal biological sequence alignment without enumerating all the possible solutions was introduced by Needleman and Wunsch [41] in 1970. This algorithm defined the similarity score between two sequences as the sum of all individual elementary similarities. The elementary similarities consist of substitutions, deletions, and insertions. At the time, Needleman and Wunsch were not aware that their algorithm belonged to a class of dynamic programming methods which had been used in speech processing and computer science, as well as other applications [33]. The solution to this class of problems can be obtained by using a sequence of smaller solutions to the same problem.

To describe the sequence analysis algorithms, let us first define the following basic notation and terminology.

M = length of sequence A
N = length of sequence B
$A_M = a_1a_2a_3 \ldots a_M$
$B_N = b_1b_2b_3 \ldots b_N$
$S_{M,N}$ = optimal similarity score between sequences A_M and B_N
$sub(a_i, b_j)$ = score for substituting b_j for a_i
$sub(a_i, -)$ = score for deleting a_i
$sub(-, b_j)$ = score for inserting b_j
$g = sub(a_i, -) = sub(-, b_j)$ = gap score
$S_{i,j}$ = the accumulative similarity score between sequences A and B up to the ith and jth positions

The original sequences A_M and B_N do not have any gaps. To align the residues between these two sequences, gaps are introduced. Let A'_L and B'_L be the two aligned sequences, where $L = MAX\{M + N_a, N + N_b\}$, N_a is the total number of gaps (number of dashes) introduced in the original sequence A_M, and N_b is the total number of gaps introduced in the original sequence B_N. A gap (-) may be introduced in and at the end of both sequences. Consider the following aligned sequences.

$$A'_L = \texttt{CCTA--GA-}$$
$$B'_L = \texttt{--TATGGAC}$$

In the above alignment, $M = 6$, $N = 7$, $L = 9$, $N_a = 3$, and $N_b = 2$. A'_L has one internal gap of length two and one external gap of length one. B'_L has one external gap of length two. The alignment score between two sequences is defined by (1). Therefore, the above alignment has a score of –1 if the identity substitution matrix is used. Usually, the external gaps are not penalized in practice. Furthermore, consecutive dashes (a single gap) are penalized by a linear function such as $w_k = u + vk$ instead of $w_k = vk$, where k is the length of the gap, u is the penalty for initiating a gap, and v is the penalty for extending a gap and is less than u. Note that $u = g = sub(a_i, -) = sub(-, b_i)$, which can be obtained from a substitution score matrix. If the external gaps are not penalized, $u = g = sub(a_i, -) = sub(-, b_i) = -1$, and $v = 0$, the score of the above alignment would be three.

$$S(A'_L, B'_L) = \sum_{i=1}^{L} sub(a_i, b_i). \tag{1}$$

The Needleman and Wunsch algorithm uses a constant gap penalty. That is, it gives the same penalty for each gap regardless of its length. In reality, we want a higher penalty for a longer gap than for a shorter one. The Needleman and Wunsch algorithm was later generalized by Smith and Waterman [44] to include this criterion. In addition, the Smith and Waterman algorithm can be used for three different types of comparisons. This algorithm is represented by (2). The gap penalty function is defined as $w_k = u + vk$, $(u, v \le 0)$, where k is the gap length, u is the penalty for initiating a gap, and v is the penalty for extending a gap.

$$S_{i,j} = MAX\begin{cases} MAX\{S_{i-kj} + w_k\} & 1 \le k \le i \\ S_{i-1,j-1} + sub(a_i, b_j) & \\ MAX\{S_{i,j-l} + w_k\} & 1 \le l \le j \end{cases} \tag{2}$$

The initial conditions for this algorithm depend on the type of comparison that we wish to perform. There are three types of comparison as described by Yap et al. [51]:

1) sequence to sequence,
2) subsequence to sequence, and
3) subsequence to subsequence.

The Smith and Waterman [44] algorithm requires $M^2 \times N$ computation steps, while the Needleman and Wunsch requires only $M \times N$. For comparing long sequences, the Smith and Waterman algorithm is computationally prohibitive. To reduce the computation steps, Gotoh [21] modified the Smith and Waterman algorithm as shown in (3) to (5), which requires only $M \times N$ steps. Note that $w_1 = u + v$, since $k = 1$.

$$S_{i,j} = MAX\begin{cases} P_{i,j} \\ S_{i-1,j-1} + sub\left(a_i, b_j\right) \\ Q_{i,j} \end{cases} \tag{3}$$

$$P_{i,j} = MAX\begin{cases} S_{i-1,j} + w_1 \\ P_{i-1,j} + v \end{cases} \tag{4}$$

$$Q_{i,j} = MAX\begin{cases} S_{i,j-1} + w_1 \\ Q_{i,j-1} + v \end{cases} \tag{5}$$

The comparison score defines the degree of similarity between two sequences. To see the similarity patterns between the two sequences, an alignment must be constructed. To obtain the alignment, we must keep track of all the paths leading to this optimal score. That is, we must save all the editing steps (substitutions, deletions, and insertions). One way to keep track of all the editing steps is to save the entire matrix S for backtracking [20].

The backtracking method is fast but it requires $O(M \times N)$ memory space, which limits its application. For example, consider the longest sequence in GenBank which has 684,973 residues. At least 1,877 Gbytes of memory are required to align this sequence against itself, assuming that an integer has four bytes. To overcome this problem, Myers and Miller [40] applied the divide and conquer algorithm of Hirschberg [26] to obtain the same alignment but require only $O(N)$ memory space. The main idea behind this method is to determine the midpoint of an optimal alignment using a forward and a reverse application of the Gotoh's algorithm. Then, the optimal alignment can be obtained by recursively determining optimal alignments on both sides of this midpoint.

The dynamic programming algorithm discussed above gives the maximum sensitivity for comparing two sequences. Sensitivity is a measure of how well a method can detect the actual similarity between two sequences. At present, there are three widely used fast heuristic algorithms for comparing two sequences. These algorithms are faster than the dynamic programming algorithm but they give lower sensitivity. Yap et al. [54] discuss and compare three programs, FASTA, BLAST, and FLASH, that use these heuristic approaches. The performance of the parallel sequence similarity searching methods that we discuss in this paper is not dependent on the sequence comparison algorithm used.

As more sequences were generated, researchers began to use multiple sequence alignment to better study the relationship between sequences. A multiple sequence alignment is defined similarly to the pairwise alignment. It is a configuration that is obtained by stacking up sequences on each other. A minimal number of gaps is introduced in the sequences so that the number of matches or similarities in each column is maximized. Mathematically, a multiple sequence alignment score is defined by (6), where n is the number of sequences and L is the length of the longest aligned sequence. A_i is the ith sequence and $A_{i,k}$ is the residue at the kth position of the ith sequence. $G(A_i)$ is a gap penalty function. Actually, $Score(A_n)$ is the sum of all possible pairwise alignment scores that can be constructed from n sequences. For n sequences, there are $\frac{(n-1)n}{2}$ possible pairwise alignment scores. The objective of multiple sequence alignment is to maximize $Score(A_n)$.

$$Score\left(A_n\right) = \sum_{k=1}^{L}\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} sub\left(A_{i,k}, A_{j,k}\right) - \sum_{i=1}^{n} G\left(A_i\right). \tag{6}$$

Many strategies have been developed to optimize this function. In most strategies, two-group alignment is used repetitively to obtain the multiple sequence alignment. In two-group alignment, the sequences from one group are aligned against those in the other group while freezing the alignment within each group. To align two groups of sequences, the algorithm of two-sequence alignment ((3) to (5)) can be extended as follows:

$$S_{i,j} = MAX\begin{cases} P_{i,j} \\ S_{i-1,j-1} + sub'\left(a_i, b_j\right) \\ Q_{i,j} \end{cases} \tag{7}$$

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

where K is the number of sequences in the X group and L in the Y group. X_i is referred to the ith position of all aligned sequences in the X group and $X_{k,i}$ is referred to a residue at the ith position of the kth sequence in the X group. $P_{i,j}$, $Q_{i,j}$, and the initial conditions remain the same. Equations (3) and (7) look the same, but the sub' function of (7) is more complicated.

3 Parallel Similarity Sequence Searching

We only present computational approaches, for the first phase, that can be used to speed up the search for the most similar sequence in the database. Only the sequence identification is retrieved. Once the identification of the similar sequence has been retrieved, we can use any of a number of available software tools to retrieve its complete entry.

There are two general parallel methods that can be used to search sequence databases. One method is to parallelize the comparison algorithm where all processors cooperate to determine each similarity score [8], [16], [34]. The other method is to parallelize the entire database comparison

process where each processor performs a selected number of comparisons independently [14], [22], [37], [43]. The first method is more suitable for the single instruction stream, multiple data stream (SIMD) parallel computer, where all the processors execute the same instruction at the same time, and the communication speed between processors is fast relative to the performance of a single processor. The second method is more suitable for the multiple instruction stream, multiple data stream (MIMD) parallel computer, where each processor is significantly more powerful than a SIMD processor, and the processors execute their instructions independently. We only evaluate methods using MIMD systems.

The second method uses a coarser grain parallelism approach, which has lower communication overhead. However, the overall performance of the coarser grain approach depends on its load balancing technique. The load balancing technique must be able to assign approximately the same amount of work to each processor. We examine and later combine two previously developed load balancing techniques. The first technique (portion method) was presented by Guan et al. [22] and the second one (master-worker) by Sittig et al. [43], Miller et al. [37], and Deshpande et al. [14]. We compare our method with the master-worker and the portion methods. This comparison is limited to the parallel computational methods, not the parallel computing system or the sequence comparison algorithms used for implementation.

To efficiently use a parallel computer system, a balanced workload among the processors is required. To compare the workload balancing effectiveness of a computational method, let the percentage of load imbalance (*PLIB*) be defined as

$$PLIB = \left\{ \frac{LargestLoad - SmallestLoad}{LargestLoad} \right\} \times 100. \quad (8)$$

PLIB is the percentage of the overall processing time that the first finished processor must wait for the last processor to finish. This number also indicates the degree of parallelism. For example, if *PLIB* is less than one, we achieve over a 99 percent degree of parallelism. Therefore, a computational method with a lower *PLIB* is more efficient than another one with a higher *PLIB*. The workload is perfectly balanced if *PLIB* is equal to zero.

To balance the workload, Guan et al. partition the database into a number of portions according to the number of processors allocated. To achieve ideal workload balance, where $PLIB = 0$, the size of each portion must be equal to the size of the database divided by the number of allocated processors. However, obtaining this ideal size for each portion is not easily attained since the database sequences have greatly varying length. The last sequence assigned to each portion does not always result in the perfect size for that portion, with the sequence being either too short or too long. If the last sequence assigned to portion P causes the sum of the sequence lengths in this portion to exceed the ideal size by more than X percent, it is reassigned to portion P + 1. This process continues until all the sequences have been assigned to one of the portions. In this method, each processor is assigned to search each portion independently. Consequently, this method has a low communication overhead, but may result in a high *PLIB* as shown below.

To minimize the *PLIB* for the Guan et al. method, the percentage value X should be set to zero. In a worst case scenario, *PLIB* can be expressed as,

$$PLIB = \frac{(1+X)Psize - \big(1-(N-1)X\big)Psize}{(1+X)Psize} \times 100, \quad (9)$$

where *Psize* is the ideal portion size, and *N* is the number of compute nodes. In this scenario, we assume that the size of every portion, except the last one, is equal to the allowable maximum: $(1 + X)Psize$. Equation (9) can be simplified into $PLIB = \frac{XN}{1+X}$, which shows that, for any nonnegative value of X, *PLIB* is minimal when X is zero.

In the Guan et al. method, the workload for each processor is determined prior to the commencement of the actual execution. In the Sittig et al. method, the workload is assigned to each processor dynamically during query processing. Sittig et al. view the querying process as a job which consists of a number of independent tasks (comparisons). They use a processor as the master to distribute the sequences to the workers (the remaining processors) for comparison and to collect the similarity scores back from them. The master's main job is to keep all the workers busy as long as there are sequences to be compared. That is, the master assigns a new sequence, as soon as possible, to the worker that completes its assigned comparison. This form of dynamic load balancing continues until all the sequences have been compared. To minimize *PLIB*, the database sequences are first sorted by their lengths in decreasing order before they are distributed by the master to the workers.

The Sittig et al. master-worker method has the opposite advantage and disadvantage from the Guan et al. portion method. That is, the master-worker method has a low *PLIB* but a high communication overhead. Although the workers need to communicate with the master only to obtain new sequences and to report new similarity scores, this method still has significantly higher communication overhead than the Guan et al. portion method. In addition, the master may not have enough work to do if there are too few workers. On the other hand, the master can become a system bottleneck if there are too many workers. If the master is overloaded, the workers' idle times are increased.

To reduce the communication overhead, eliminate the potential bottleneck from having a master processor, and obtain a low *PLIB*, we combine the advantages from both the Guan et al. and the Sittig et al. methods. In the combined (bucket) method, a greedy allocation method (as found in Sittig et al.) is performed statically (as in Guan et al.). In the bucket method, the original sequences in the database are placed into one of the N buckets (smaller databases), where N is the largest number of possible processors, so that the difference between the sum of the sequence lengths in the smallest and the largest buckets is minimized. That is, the *PLIB* is minimized.

The algorithm for placing the database sequences in these buckets is as follows: First, the sequences are sorted in decreasing length order. Then, starting from the longest one, each sequence is placed into the bucket that has the

TABLE 1
PERCENTAGE OF LOAD IMBALANCE FOR VARYING NUMBER OF PROCESSORS

No. of Processors	*Portion* Method			*Bucket* Method
	Unsorted	Ascending Sorted	Descending Sorted	Descending Sorted
2	0.00189	0.00246	0.00150	0.00000
4	0.13492	0.00587	0.01239	0.00000
8	0.28340	0.04476	0.03199	0.00001
16	0.60677	0.30807	0.56720	0.00003
32	5.23125	2.56729	1.50670	0.00007
64	9.68066	10.17033	6.76098	0.00011
128	15.52223	28.67652	23.93645	0.00044

TABLE 2
SERIAL SEARCH TIMES IN SECONDS FOR SIX QUERY SEQUENCES OF DIFFERENT LENGTHS

Seq Length	50	100	200	400	800	1,600
Time (sec)	16,357	32,342	66,389	138,510	300,125	606,162

current smallest sum of sequence lengths. In the case of a tie, the smallest numbered bucket is selected. This, in fact, is the static equivalency of the dynamic method of Sittig et al. As in the Guan et al. method, each processor can search its own bucket independently without communicating with other processors. If only $\frac{N}{n}$ processors are used, each processor searches n buckets.

The bucket and the Guan et al. portion methods can be compared analytically based on the *PLIB* since the only difference between the two methods is the workload on each processor. Therefore, we compared the two methods analytically based on *PLIB* using the entire GenBank (release 86.0) database. The *PLIB* is calculated for a varying number of processors based on our formula (8). The variables, *LargestLoad* and *SmallestLoad*, in (8) are replaced by *LargestBucket* (*LargestPortion*) and *SmallestBucket* (*SmallestPortion*), respectively. The percentages of load imbalance for both methods are shown in Table 1. The results in this table show that the bucket method produced a smaller *PLIB* than the portion method in all cases. For the portion method, we calculated the *PLIB* for unsorted, ascending sorted, and descending sorted databases to determine which one gives the best performance. The database is sorted according to its sequence lengths.

The main difference between the portion and the bucket methods is the strategy used to decompose the domain of the input data. In both methods, all processors execute the same algorithm, but the data streams are different. Consequently, we can easily compare the performance of these two methods analytically based on the workload decomposition. Unlike the portion and the bucket methods, the master-worker method used a control decomposition strategy which does not predetermine the workload for each processor. Therefore, we cannot use the percentage of load imbalance to compare the bucket and the master-worker methods analytically.

To compare the bucket method with the master-worker method, we implemented both of them on the Intel iPSC/860 parallel computer. We then obtained run times for six query sequence lengths using the entire GenBank (release 86). Starting with the first sequence of 50 bases, the lengths of the subsequent query sequences were repeatedly increased by a factor of two. In the following tables, we identify these sequences by their lengths. For example, the query sequence with 50 bases is referred to as sequence 50.

The serial search times in seconds for the six query sequences are shown in Table 2. These serial search times are obtained by executing a serial program on a single processor and are used to calculate the speedup factors for the bucket and the master-worker methods. The speedup factors for the bucket and the master-worker methods based on the implementation on the Intel iPSC/860, as shown in Table 3 and Table 4, were calculated for all combinations of the number of processors and the six query sequences. The speedup factor is the ratio of the total run time of the serial version of the program to the total run time of the parallel version of the program. Table 3 and Table 4 show that the bucket method also performed better than the master-worker method in all cases.

The computation times used to calculate the speedup factors included the I/O time, the computation time, and the communication time required for each search. However, it did not include the preprocessing time involved in sorting and placing the database sequences into the 128 buckets as this task was performed only once. For the current release of GenBank (release 86.0), it took the host machine (SPARC station 2) only 80 seconds to sort the sequence indexes and 31 seconds to place these indexes into the 128 buckets. Thus, for total fairness, we should include an overhead of 111 seconds divided by the number of runs per GenBank release. GenBank is usually updated every two months.

The results in Table 3 show that the bucket method performed consistently better than the master-worker method in all cases. The bucket method achieved a near ideal speedup for all six query sequences, except the shortest one. The master-worker method achieved significantly lower speedups than the ideal one when a large number of processors was used, as seen in Table 4. This was due to the fact that the master was overloaded with a large number of worker processors. Consequently, we expect the bucket method to perform significantly better than the master-worker method on a system that has a larger number of processors (more than 128). If we keep increasing the number

TABLE 3
THE SPEEDUP FACTORS OF THE BUCKET METHOD

	Query Sequence Lengths					
N	50	100	200	400	800	1,600
2	2.00	1.99	1.97	2.00	2.00	2.00
4	3.99	3.97	3.94	3.99	3.99	4.00
8	7.96	7.98	7.96	7.99	7.98	8.00
16	15.95	15.96	15.92	15.98	15.98	16.00
32	30.93	31.57	31.81	31.96	31.97	31.99
64	61.68	63.33	63.09	63.80	63.93	63.96
128	122.05	126.73	126.94	127.69	127.71	127.76

TABLE 4
THE SPEEDUP FACTORS OF THE MANAGER-WORKER METHOD

	Query Sequence Lengths					
N	50	100	200	400	800	1,600
2	1.00	1.00	1.00	1.00	1.00	1.00
4	2.98	2.99	2.99	3.00	3.00	3.00
8	6.81	6.95	6.95	6.99	7.00	7.00
16	13.69	14.61	14.73	14.99	14.99	15.00
32	22.06	28.07	30.34	30.91	30.95	30.98
64	28.62	45.18	57.41	62.01	62.71	62.90
128	30.16	57.21	91.90	117.22	124.92	126.32

of processors, the master would eventually become a system bottleneck (heavily overloaded), particularly when searching using short sequences. This would not be a problem for the bucket method provided that the number of sequences in the database also increases. Since the number of sequences in the GenBank is actually growing, the degree of scalability of the bucket method will become of even greater importance. Scalability is the ability to maintain good performance when the number of processors is increased.

For short query sequences, the master method achieved significantly lower speedups than the bucket method. This is due to the fact that the master could not serve a large number of workers efficiently when a given query sequence was short. As the workload assigned to the master was too great, the workers finished the tasks faster than the master could distribute them. Thus, some workers had to wait idly to receive another task from the master while it was serving other workers. The master became the bottleneck when greater than 16 processors were used. This is a serious drawback, since, sometimes, query sequences are very short. For example, recently, biologists designed DNA primers with 30 to 50 bases to block Alu repetitive sequences from amplifying during polymerase chain reaction.

We discuss only the parallel computational methods that compare a query sequence with all the sequences in the database using the full dynamic programming algorithm. To reduce the execution time, some researchers use heuristic techniques to reduce the number of database sequences before they apply the full dynamic programming algorithm. Both Sittig et al. and Guan et al. showed that they could reduce the search time by applying some heuristic techniques. However, it is difficult to evaluate and compare these heuristic techniques because they do not produce the same set of answers, and no qualitative assessment of the near optimality of the heuristic technique is available.

4 PARALLEL MULTIPLE SEQUENCE ALIGNMENT

A multiple sequence alignment is a configuration that is obtained by stacking up sequences on top of each other. A minimal number of gaps is introduced in the sequences so that the number of matches is maximized according to a given optimization function. The rigorous pairwise alignment algorithms [21], [41], [44], as described in Section 2, can be extended to align more than two sequences. However, it is impractical to extend this rigorous algorithm to align more than three sequences, since memory space and computation time is proportional to the product of the sequence lengths. These algorithms have been extended to align only three sequences [38], [39].

To practically align a large number of sequences, many researchers use the tree-based methods which generate multiple sequence alignments by aligning and combining a number of pairwise alignments in a particular order. For a simple tree, where there is only one branch at each level, the order is linear [39], [36], [46]. These linear ordering strategies start with the most similar sequence pair and continue to add sequences in decreasing order of 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 [47], these strategies can produce a poor alignment if the sequences belong to two or more distinct subfamilies. To avoid this situation, sophisticated ordering strategies [3], [11], [18], [23], [24], [46] were introduced. These strategies apply various clustering techniques to order groups of related sequences in a hierarchical tree. Then, the final multiple sequence alignment is obtained by aligning and combining clusters of sequences, proceeding from the leaves to the root of the tree.

The order in which the sequences are aligned and combined 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 alignment. However, a few researchers [6], [29], [32], [35] took an opposite direction by not using any ordering. Instead, they applied randomized techniques with optimization functions to iteratively improve the multiple sequence alignment. The iterative methods have been shown to produce better alignments than the tree-based methods [25]. In addition, the randomized iterative algorithms can be used to improve the alignment that were generated from other algorithms. In this case, they can never do worse. However, they require a much longer computation time. In this paper, we use the Berger-Munson algorithm [6] to illustrate that the computation times of these methods can be reduced significantly by using a parallel speculative computation technique.

Fig. 1 shows the core part of the Berger-Munson algorithm. The C language implementation contains approximately 1,900 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

```
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);
}
```

Fig. 1. A Berger-Munson sequential algorithm.

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.

We adapted the restricted search space, as presented by Ishikawa et al. [27], [28], 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. We discovered that the reason was due to the fact that the similarity among the sequences in one group decreases if more than two sequences are allowed in that group. Thus, the alignment between the two groups of sequences also has a lower similarity score. 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 $2^{n-1} - 1$. Therefore, $q = n + \frac{n(n-1)}{2}$.

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 ith 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.

This algorithm was previously parallelized by Ishikawa et al. [27], [28] on a parallel inference machine (PIM) using a parallel logic programming language (KL1). However, as pointed out by Yap et al. [53], their parallel version has a few drawbacks. First, this version becomes impractical for aligning a large number of sequences. 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.

Speculative computation [45] has been applied efficiently to parallelize sequential algorithms, such as 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 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 Fig. 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 ith iteration may depend on the $(i - 1)$th iteration. To be exact, the ith 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 Fig. 2, the first 28 sequential iterations can be reduced to 13 parallel steps if four processors are used. The parallel computation steps are shown in Fig. 3. Three iterations $(p - 1)$ are speculated at each parallel step, where P1 speculates that P0 will reject its new alignment, P2 speculates that P0 and P1 will reject their new alignments, and P3 speculates that P0 to P2 will reject their new alignments. P0 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

Sequential Iteration number	1234567...
Decision sequence	AAAAARAAARRARRRARRRARRRARRRRR...

Fig. 2. A possible sequential decision sequence.

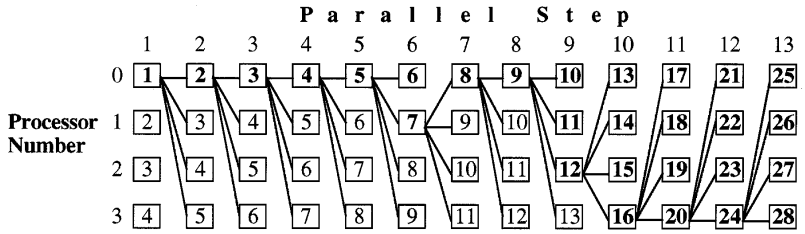

Fig. 3. An illustration of the parallel speculative computation process.

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 a parallel implementation. Our parallel algorithm, which is implemented on every processor, is summarized in Fig. 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 interprocessor data transfer is much faster than the I/O data transfer. Initially, every new alignment is usually accepted. Consequently, 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 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 to P–1){
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;
29  }
```

Fig. 4. A 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. 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 interprocessor 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 *i*th bit is set when the *i*th 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 as 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).

To evaluate the performance of our approach, we have used it to improve the alignments generated manually by experts, Kabat et al. [30], and automatically by a popular multiple sequence alignment program, CLUSTALV [17], [18], 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 [30]. Their statistical summaries are shown in Table 5. 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 5
STATISTICAL SUMMARIES OF THE THREE GROUPS OF TEST SEQUENCES

Group Name	Num. of Seqs	Avg. Length	Initial Score
CLLC	93	62	1,574,724
HHC3	185	65	66,50,831
MKL5	324	83	31,393,504

The scores of the alignments manually generated by experts, Kabat et al. [30] and their improved scores performed by the sequential Berger-Munson program, MUSEQAL, are shown in Table 6. The number of iterations and the sequential computation times taken by MUSEQAL are also shown in this table. These sequential computation times were obtained by 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 7. Comparing Table 6 and Table 7, 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 8 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 best sequential solution of the program to the total run time of the parallel version. Table 8 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. Consequently, there were more rejections in improving the Kabat alignments than the CLUSTALV alignments.

It is difficult to accurately compare our speedup factors with those obtained by Ishikawa et al. [27], [28], since their parallel algorithm does not always generate the same alignment as the sequential one. To evaluate their parallel algorithm, they aligned seven sequences which have 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 10. 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.

We used the MUSEQAL program to improve the alignments generated by CLUSTALV and by an expert by improving the alignment score. A better alignment score only means better alignment. It does not necessarily mean that

TABLE 6
KABAT AND MUSEQAL ALIGNMENT SCORE COMPARISON

Group Name	Kabat Score	MUSEQAL Score	No. of Iteration	Serial Time (s)
CLLC	1,826,747	1,956,501	16,193	54,708
HHC3	7,505,258	7,681,308	56,232	566,901
MKL5	38,568,971	38,765,838	112,374	2,534,786

TABLE 7
CLUSTALV AND MUSEQAL ALIGNMENT SCORE COMPARISON

Group Name	CLUSTALV Score	MUSEQAL Score	No. of Iteration	Serial Time (s)
CLLC	1,809,065	1,956,771	12,716	35,871
HHC3	7,366,275	7,655,223	58,285	564,013
MKL5	37,778,278	38,749,102	112,390	2,611,205

TABLE 8
PARALLEL SPECULATIVE BERGER–MUNSON ALGORITHM SPEEDUP FACTORS

	Speedup w.r.t Kabat Alignment			Speedup w.r.t CLUSTAL Alignment		
No. of Proc.	CLLC	HHC3	MKL5	CLLC	HHC3	MKL5
1	1.00	1.00	1.00	1.00	1.00	1.00
2	1.84	1.86	1.93	1.78	1.81	1.89
4	3.38	3.40	3.75	3.26	3.27	3.65
8	6.41	6.22	7.27	6.12	5.92	7.05
16	11.56	11.39	14.10	10.70	10.72	13.61
32	19.46	20.84	28.28	16.98	19.40	26.26
64	29.52	38.14	53.06	23.75	35.12	50.68

the sequences have a higher biological similarity or they are more related biologically. In most cases, however, better alignment scores do indicate more biological similarity.

5 CONCLUSION

We presented and evaluated two parallel computational methods for analyzing biological sequences. These methods allow researchers to analyze biological sequences at a much higher speed than the sequential methods. In addition, they also make it possible for scientists to analyze problems that were previously considered too large. The first method is used to retrieve information about the sequences that are similar to the query sequences. We showed that this method achieved a near perfect speedup. Generally, only a number of N most similar sequences are retrieved. To analyze these similar sequences further, we developed another parallel computational method for aligning these N sequences. This second problem is highly sequential and was difficult to parallelize. In fact, the only prior parallization of this biologically significant computation achieved a speedup factor of only 10 using 64 nodes. However, we showed that our method was able to achieve speedup factors ranging from 23 to 53 on a 64 nodes.

Sequence searches are very important to the biological research community. These searches can save research time and lead to new discoveries. For example, a laboratory experiment for determining a particular property or function of a sequence can be avoided if that experiment has already been done by another researcher and its results are stored in one of the sequence databases. Sequence searches have also led to the discovery of many protein families, including the tyrosine kinase oncogene, the steroid receptor, and the transcription factors containing a zinc-finger motif. In addition, they also provide insights into the mechanisms of actions of newly discovered sequences.

Multiple sequence alignment has been shown to be an important method in studying the structural and functional properties of proteins and the organism in which they are expressed. It has been used to determine common ancestors of sequences and the organisms in which these are expressed, to classify organisms, and to reveal structurally and functionally important regions of proteins, such as catalytic sites and ligand binding sites. This tool can also be used to help to predict the structure of proteins.

ACKNOWLEDGMENTS

The research of Ophir Frieder was supported, in part, by the U.S. National Science Foundation under contract #IRI-9357785.

REFERENCES

[1] A. Bairoch and B. Boeckmann, "The SWISS-PROT Protein Sequence Data Bank," *Nucleic Acids Research*, vol. 20, pp. 2,019-2,022, 1992.
[2] G.J. Barton and M.J.E. Sternberg, "A Strategy for the Rapid Multiple Alignment of Protein Sequences," *J. Molecular Biology*, vol. 198, pp. 327-337, 1987.
[3] G.J. Barton, "Protein Multiple Sequence Alignment and Flexible Pattern Matching," *Methods in Enzymology*, vol. 183, pp. 403-427, 1990.
[4] G.J. Barton, "Scanning Protein Sequence Databanks Using a Distributed Processing Workstation Network," *Computer Applications in the Biosciences*, vol. 7, pp. 85-88, 1991.
[5] D. Benson, D.J. Lipman, and J. Ostell, "GenBank," *Nucleic Acids Research*, vol. 21, pp. 2,963-2,965, 1993.
[6] M.P. Berger and P.J. Munson, "A Novel Randomized Iteration Strategy for Aligning Multiple Protein Sequences," *Computer Applications in the Biosciences*, vol. 7, pp. 479-484, 1991.
[7] F.C. Bernstein, T.F. Koetzle, G.J.B. Williams, E.F. Meyer Jr., M.D. Brice, J.R. Rodgers, O. Kennard, T. Shimanouchi, and M. Tasumi, "The Protein Data Bank: A Computer-based Archival File for Macromolecular Structures," *J. Molecular Biology*, vol. 112, pp. 535-542, 1977.
[8] D.L. Brutlag, J.P. Dautricourt, R. Diaz, F. Fier, B. Moxon, and R. Stamm, "BLAZE: An Implementation of the Smith-Waterman Sequence Comparison Algorithm on a Massively Parallel Computer," *Computers Chem.*, vol. 17, pp. 203-207, 1993.
[9] C. Burks, M. Cassidy, M.J. Cinkowsky, K.E. Cumella, P. Gilna, J.E.H. Hayden, G.M. Keen, T.A. Kelly, M. Kelly, D. Kristofferson, and J. Ryals, "GenBank," *Nucleic Acids Research*, vol. 19 supplement, pp. 2,221-2,225, 1991.
[10] U.S. Congress Office of Technology Assessment, "Mapping Our Genes—The Genome Projects: How Big, How Fast?" OTA-BA-373, Washington, D.C., Government Printing Office, Apr. 1988.
[11] F. Corpet, "Multiple Sequence Alignment with Hierarchical Clustering," *Nucleic Acids Research*, vol. 16, no. 22, pp. 10,881-10,891, 1988.
[12] M.O. Dayhoff, R.M. Schwartz, and B.C. Orcutt, "A Model of Evolutionary Change in Proteins," *Atlas of Protein Sequence and Structure*, vol. 5, pp. 345-352, 1978.
[13] C. DeLisi, "The Human Genome Project," *American Scientist*, vol. 76, pp. 488-493, 1988.
[14] A.S. Deshpande, D.S. Richards, and W.R. Pearson, "A Platform for Biological Sequence Comparison on Parallel Computers," *Computer Applications in the Biosciences*, vol. 7, pp. 237-247, 1991.
[15] DNA Data Bank of Japan Nat'l Inst. of Genetics, Yata, Mishima, 411, Japan, ftp address: ddbj@ddbj.nig.ac.jp.
[16] E.W. Edmiston, N.G. Core, J.H. Saltz, and R.M. Smith, "Parallel Processing of Biological Sequence Comparison Algorithms," *Int'l J. Parallel Programming*, vol. 17, pp. 259-275, 1988.

[17] European Molecular Biology Laboratory, Postfach 10.2209, D-6900 Heidelberg, Federal Republic of Germany, E-mail: DataLib@EMBL-Heidelberg.DE.
[18] D.F. Feng and R.F. Doolittle, "Progressive Alignment and Phylogenetic Tree Construction of Protein Sequences," *Methods in Enzymology*, vol. 183, pp. 375-387, 1990.
[19] D.G. George, W.C. Barker, and L.T. Hunt, "The Protein Identification Resource (PIR)," *Nucleic Acids Research.*, vol. 14, pp. 11-15, 1986.
[20] D.G. George, L.T. Hunt, and W.C. Barker, "Current Methods in Sequence Comparison and Analysis," *Macromolecular Sequencing and Synthesis Selected Methods and Applications*, pp. 127-149, 1988.
[21] O. Gotoh, "An Improved Algorithm for Matching Biological Sequences," *J. Molecular Biology*, vol. 162, pp. 705-708, 1982.
[22] X. Guan, R. Mural, R. Mann, and E. Uberbacher, "On Parallel Search of DNA Sequence Databases," *Proc. Fifth SIAM Conf. Parallel Processing for Scientific Computing*, pp. 332-337, 1991.
[23] D.G. Higgins and P.M. Sharp, "CLUSTAL: A Package for Performing Multiple Sequence Alignment on a Microcomputer," *Gene*, vol. 73, pp. 237-244, 1988.
[24] D.G. Higgins and P.M. Sharp, "Fast and Sensitive Multiple Sequence Alignments on a Microcomputer," *Computer Applications in the Biosciences*, vol. 5, pp. 151-153, 1989.
[25] M. Hirosawa, Y. Totoki, M. Hoshida, and M. Ishikawa, "Comprehensive Study on Iterative Algorithms of Multiple Sequence Alignment," *Computer Applications in the Biosciences*, vol. 11, pp. 13-18, 1995.
[26] D.S. Hirschberg, "A Linear Space Algorithm for Computing Maximal Common Subsequences," *Comm. ACM*, vol. 18, pp. 341-343, 1975.
[27] M. Ishikawa, M. Hoshida, M. Hirosawa, T. Toya, O. Kentaro, and K. Nitta, "Protein Sequence Analysis Program: Multiple Sequence Alignment by Parallel Iterative Aligner," *Demonstrations Int'l Conf. Fifth Generation Computer Systems*, Tokyo, pp. 57-62, 1992.
[28] M. Ishikawa, M. Hoshida, M. Hirosawa, T. Toya, O. Kentaro, and K. Nitta, "Protein Sequence Analysis Program by Parallel Inference Machine," *Proc. Int'l Conf. Fifth Generation Computer Systems*, Tokyo, pp. 294-299, 1992.
[29] M. Ishikawa, T. Toya, M. Hoshida, K. Nitta, A. Ogiwara, and M. Kanehisa, "Multiple Sequence Alignment by Parallel Simulated Annealing," *Computer Applications in the Biosciences*, vol. 9, pp. 267-273, 1993.
[30] 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, Nat'l Inst. of Health, NIH Publication No. 91-3242, 1991.
[31] G. Keen, G. Redgrave, J. Lawton, M. Cinkowsky, S. Mishra, J. Fickett, and C. Burks, "Access to Molecular Biology Databases," *Mathematical Computer Modeling*, vol. 16, pp. 93-101, 1992. Internet: limb@life.lanl.gov. To obtain the LiMB file, send the message: limb-data to bioserve@t10.Lanl.GOV.
[32] J. Kim, S. Pramanik, and M.J. Chung, "Multiple Sequence Alignment Using Simulated Annealing," *Computer Applications in the Biosciences*, vol. 10, pp. 419-426, 1994.
[33] J.B. Kruskal, "An Overview of Sequence Comparison: Time Warps, String Edits, and Macromolecules," *SIAM Review*, vol. 25, pp. 201-237, 1983.
[34] E. Lander, J.P. Mesirov, and W. Taylor IV, "Study of Protein Sequence Comparison Metrics on the Connection Machine CM-2," *J. Supercomputing*, vol. 3, pp. 255-269, 1989.
[35] C.L. Lawrence, S.F. Altschul, M.S. Boguski, J.S. Liu, A.F. Neuwald, and J.C. Wootton, "Detecting Subtle Sequence Signals: A Gibbs Sampling Strategy for Multiple Alignment," *Science*, vol. 262, pp. 208-214, 1993.
[36] H.M. Martinez, "A Flexible Multiple Sequence Alignment Program," *Nucleic Acids Research*, vol. 16, pp. 1,683-1,691, 1988.
[37] P.L. Miller, P.M. Nadkarni, and W.R. Pearson, "Comparing Machine-Independent versus Machine-Specific Parallelization of a Software Platform for Biological Sequence Comparison," *Computer Applications in the Biosciences*, vol. 8, pp. 167-175, 1992.
[38] M. Murata, J.S. Richardson, and J.L. Sussman, "Simultaneous Comparison of Three Protein Sequences," *Proc. Nat'l Academy of Sciences USA*, vol. 82, pp. 3,073-3,077, 1985.
[39] M. Murata, "Three-Way Needleman-Wunsch Algorithm," *Methods in Enzymology*, vol. 183, pp. 365-375, 1990.
[40] E.W. Myers and W. Miller, "Optimal Alignments in Linear Space," *Computer Applications in the Biosciences*, vol. 4, pp. 11-17, 1988.
[41] S.B. Needleman and C.D. Wunsch, "A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Sequences," J. *Molecular Biology*, vol. 48, pp. 443-453, 1970.
[42] Protein Information Resource Nat'l Biomedical Research Foundation, 3900 Reservoir Road, N.W., Washington, D.C. 20007, E-mail: PIRMAIL@GUNBRF.bitnet.
[43] D.F. Sittig, D. Foulser, N. Carriero, G. McCorkle, and P.L. Miller, "A Parallel Computing Approach to Genetic Sequence Comparison: The Master-Worker Paradigm with Interworker Communication," *Computers and Biomedical Research*, vol. 24, pp. 152-169, 1991.
[44] T.F. Smith and M.S. Waterman, "Identification of Common Molecular Subsequence," *J. Molecular Biology*, vol. 147, pp. 195-197, 1981.
[45] A. Sohn, "Parallel N-ary Speculative Computation of Simulated Annealing," *IEEE Trans. Parallel and Distributed Systems*, vol. 6, pp. 997-1005, 1995.
[46] W.R. Taylor, "Multiple Sequence Alignment by a Pairwise Algorithm," *Computer Applications in the Biosciences*, vol. 3, pp. 81-87, 1987.
[47] W.R. Taylor, "A Flexible Method to Align Large Numbers of Biological Sequences," *J. Molecular Evolution*, vol. 28, pp. 161-169, 1988.
[48] V. Veljkovic, R. Meltlas, J. Raspopovic, and S. Pongor, "Spectral and Sequence Similarity between Vasoactive Intestinal Peptide and the Second Conserved Region of Human Immunodeficiency Virus Type 1 Envelope Glycoprotein (gp120): Possible Consequences on Prevention and Therapy of AIDS," *Biochemical and Biophysical Research Comm.*, vol. 189, pp. 705-710, 1992.
[49] M.A. Watson and T.P. Fleming, "Isolation of Differentially Expressed Sequence Tags from Human Breast Cancer," *Cancer Research*, vol. 54, pp. 4,598-4,602, 1994.
[50] E.E. Witte, R.D. Chamberlain, and M.A. Franklin, "Parallel Simulated Annealing Using Speculative Computation," *IEEE Trans. Parallel and Distributed Systems*, vol. 2, pp. 483-494, 1991.
[51] T.K. Yap, O. Frieder, and R.L. Martino, "Parallel Computation in Biomedicine: Genetic and Protein Sequence Analysis," *Handbook of Parallel and Distributing Computing*, A.Y. Zomaya, ed., pp. 1,071-1,096. McGraw-Hill, 1996.
[52] T.K. Yap, O. Frieder, and R.L. Martino, "Parallel Homologous Sequence Searching in Large Databases," *Proc. IEEE Fifth Symp. Frontiers of Massively Parallel Computation*, pp. 231-237, Feb. 1995.
[53] T.K. Yap, P.J. Munson, O. Frieder, and R.L. Martino, "Parallel Multiple Sequence Alignment Using Speculative Computation," *Proc. Int'l Conf. Parallel Processing*, Aug. 1995.
[54] T.K. Yap, O. Frieder, and R.L. Martino, *High Performance Computational Methods for Biological Sequence Analysis*. Kluwer Academic Publishers, 1996.

Tieng K. Yap received his BSEE (1988) from the Illinois Institute of Technology, an MSEE (1991) from Johns Hopkins University, and a PhD (1995) in information technology from George Mason University. He is currently a senior computer scientist in the Division of Computer Research and Technology (DCRT) of the National Institutes of Health (NIH). Dr.Yap is also an adjunct faculty member of the Whiting School of Engineering at Johns Hopkins University. His research interests include the application of computers in medicine and biology, parallel computing, and database applications. He is a member of Tau Beta Pi and Eta Kappa Nu.

Ophir Frieder received his BSc (1984) in computer and communications science and his MSc (1985) and PhD (1987) in computer science and engineering, all from the University of Michigan.

From 1987 to 1990, Dr. Frieder was a member of the technical staff in the Applied Research Area of Bell Communications Research. His research focused on the design and development of parallel and distributed systems aimed at improving modern-day telecommunication systems.

In 1990, Dr. Frieder joined George Mason University as an assistant professor of computer science where, since 1995, he has been a professor. In 1994-1995, he served as the associate department chair of computer science. In 1996, Dr. Frieder went on leave from George Mason and joined the Florida Technology Instutute as the Harris Professor of Computer Science. In 1997, Dr. Frieder also joined the computer engineering faculty and is now the Harris Professor of Computer Science and Computer Engineering.

Dr. Frieder has consulted for a variety of organizations. Among his various long-term consulting roles, Dr. Frieder served as a staff consultant at the Federal Bureau of Investigations (1991-1993), at the Institute for Defense Analysis (1992-1993), at IBM/Loral Federal Systems (1993-1994), and at SAIC (1994-1995). Since 1995, he has been a staff consultant at the Software Productivity Consortium.

Dr. Frieder has authored three books, published more than 80 refereed publications, was granted two patents, and has received research support from a variety of governmental and industrial grants. In 1993, he was a recipient of the International Information Science Foundation Award from Japan and the U.S. National Science Foundation National Young Investigator Award. In 1996, he was named a finalist for the George Mason University University-Wide Teaching Award.

Dr. Frieder's research interests include parallel and distributed information retrieval systems, communication systems, and biological and medical data processing architectures. Since 1995, Dr. Frieder has been on the editorial board of *IEEE Software*, where, in 1995-1996, he served as associate editor-in-chief. Currently, Dr. Frieder is also on the editorial board of *Information Retrieval*. Dr. Frieder is a member of Phi Beta Kappa and ACM, and a senior member of the IEEE.

Robert L. Martino received a BS from Northeastern University, Boston, and an MS and a PhD from the University of Maryland, College Park, all in electrical engineering. He is presently an associate director of the Division of Computer Research and Technology (DCRT) of the National Institutes of Health (NIH), as well as chief of the Computational Bioscience and Engineering Laboratory of DCRT. He is also the NIH representative to the national multiagency Computing, Information, and Communications Research and Development (CIC R&D) Program. Dr. Martino is an adjunct faculty member at the Whiting School of Engineering of Johns Hopkins University. His research interests include the application of computers in medicine and biology, parallel computing, methods for processing physiological signals and medical images, computer architecture, and digital system theory. He is a member of the IEEE, the IEEE Computer Society, and the IEEE Engineering in Medicine and Biology Society.