Efficient Parallel Algorithm for Optimal Three-Sequences Alignment

Chun Yuan Lin[1], Chen Tai Huang, Yeh-Ching Chung, and Chuan Yi Tang
Department of Computer Science, NTHU
{[1]*cyulin@mx, g936325@oz, ychung@cs, cytang@cs*}*.nthu.edu.tw*

Abstract

Sequence alignment is a fundamental problem in the computational biology. Many alignment methods have been proposed in the literature, such as pair-wise sequence alignment (2SA), syntenic alignment, multiple sequence alignment (MSA) and constraint multiple sequence alignment, etc. Three-sequence alignment (3SA) problem has been proposed and discussed in the computational biology and proved that the alignment results from 3SA are better than those from 2SA under some conditions. However, 3SA problem is less discussed over the past decade due to the computer capability. 3SA problem now is worthy to discuss due to the powerful computer and more and more genome and protein sequences. In this paper, an efficient parallel algorithm (P3SA) is proposed to solve 3SA problem. The P3SA method requires $O(n^2/p)$ space complexity and $O(n^3/p)$ time complexity. The experimental results show that P3SA algorithm is applicable and achieves a satisfied speed-up.

Index Terms- *dynamic programming, computational biology, sequence alignment, Hirschberg's technique, time and space complexities.*

1. Introduction

Sequence alignment is a fundamental problem in the computational biology [7]. Many alignment methods have been proposed in the literature, such as pair-wise sequence alignment (2*SA*, [18, 20, 23]), multiple sequence alignment (*MSA*, [4, 5, 17, 19, 25]), syntenic alignment [6], and constraint multiple sequence alignment [24], etc. The 2*SA* method typically used the dynamic programming scheme in which one or multiple tables are filled through a scoring mechanism. Once the best score in the tables is found a trace back procedure is involved for finding the optimal alignment. Let n be the maximum length of two sequences aligned. Several tables with the size of $(n+1)\times(n+1)$ are filled to find an optimal path for 2*SA*. It takes both $O(n^2)$ time and space complexities. Myers and Miller [15] applied the divide-and-conquer technique of Hirschberg [8] to reduce the space requirement to the linear space complexity.

Three-sequence alignment (3*SA*) problem has been proposed and discussed in the computational biology [1, 9, 13, 14, 16, 21]. Some methods and definitions have been presented and proved that the alignment results from 3*SA* are better than those from 2*SA* under some conditions. 3*SA* method also can be solved both in $O(n^3)$ time and space complexity by using a dynamic programming scheme [1], where n is the maximum length of these three sequences be aligned. Huang [13] extended the Myers and Miller's algorithm [15] for 2*SA* method to 3*SA* method. The 3*SA* method was done by filling several tables with the size of $(n+1)\times(n+1)\times(n+1)$. 3*SA* method can be solved in $O(n^3)$ time complexity and $O(n^2)$ space complexity. However, 3*SA* problem is less discussed over the past decade due to the computer capability. 3*SA* problem now is worthy to discuss due to the powerful computer and more and more genome and protein sequences released. Although the space requirement is reduced to quadratic space, the time complexity of 3*SA* method still limits its applicability. Hence, to reduce the time complexity of 3*SA* method becomes an important issue.

In this paper, an efficient parallel algorithm (*P*3*SA*) for 3*SA* problem is proposed to reduce the time complexity. We adopt the definitions of 3*SA* problem used in the Huang's algorithm and the divide-and-conquer technique of Hirschberg [10] to extend the 3*SA* method to *P*3*SA* method. The *P*3*SA* method requires $O(n^2/p)$ space complexity and $O(n^3/p)$ time complexity, where p is number of processors and less than n. Both theoretical analysis and experimental tests are presented. The experimental results show that a

[*]The work of this paper was partially supported by NSC under contract NSC94-2745-P-007-001 and NSC95-2745-P-007-001. [1]Corresponding author.

good performance and a satisfied speed-up are achieved by *P*3*SA* method.

The rest of the paper is organized as follows: In Section 2, a brief survey of related work is presented. Section 3 describes the definitions and algorithms of 3*SA* and *P*3*SA* methods. In Section 4, theoretical analysis of *P*3*SA* method is given. In Section 5, the experimental tests are presented.

2. Related Work

Many schemes for reducing the complexities of alignment methods have been proposed. Hirschberg [10] first proposed a linear space algorithm for computing longest common subsequences problem. Myers and Miller [15] applied the Hirschberg's technique to Gotoh's algorithm [8]. After applying the Hirschberg's technique [10], the space complexity of 2*SA* method is reduced from $O(n^2)$ to $O(n)$ and introduces a small constant (about 2) slowdown to $O(n^2)$ time complexity. Huang [12] extended the above algorithm to local sequence alignment problem. Since the size of biological sequences could be very large, these algorithms are very important that make 2*SA* method applicable. Although space-optimal algorithms reduce the space requirement for large sequence alignment, it still is a time-consuming problem. Some parallel algorithms to reduce the computational cost have been proposed. Huang [11] presented a *P*2*SA* algorithm that uses optimal $O((m+n)/p)$ space complexity and suboptimal $O((m+n)^2/p)$ time complexity, where m and n are the lengths of two sequences aligned. Aluru *et al.* [2] presented another parallel algorithm with optimal $O((mn)/p)$ time complexity but uses $O(m+(n/p))$ space complexity. Afterward a space and time optimal, $O((m+n)/p)$ space complexity and $O((mn)/p)$ time complexity, *P*2*SA* algorithm was presented by Rajko and Aluru [20]. Huang [13] extended the Myers and Miller's algorithm [15] to optimal 3*SA* with affine gap penalties. The 3SA algorithm simultaneously aligns three sequences by using a dynamic programming approach to find an optimal path in $O(n^2)$ space complexity and $O(n^3)$ time complexity.

3. Method

In this section, 3*SA* problem will be formalized first. Then a dynamic programming algorithm with a divide-and-conquer technique [10] for solving 3*SA* problem which proposed by Huang [13] will be introduced. Finally, the *P*3*SA* method will be presented.

```
A = GHBVADTHFASDFDAE
B = GSBVADTFASDAEE
C = GHBDTTHFASDDAEE

Alignment of A, B and C:

G-HBVADT-HFASDFDAE-
GS-BVADT--FASD--AEE
G-HB--DTTHFASD-DAEE
```

Figure 1. An example of the result for aligning sequences *A*, *B* and *C*.

3.1. 3*SA* problem

Let $A = a_1, a_2, \ldots, a_m$, $B = b_1, b_2, \ldots, b_n$ and $C = c_1, c_2, \ldots, c_l$ be three sequences over an alphabet Σ. Let '-' be a unique symbol not in Σ, denoted as a gap. Some definitions [13] are shown in the following.

Definition 1. *An aligned triple consists of three ordered elements in* $\Sigma \cup \{-\}$.

Definition 2. *An alignment of three sequences A*, *B* and *C is a finite sequence of aligned triples.*

For each triple, the first element is always from sequence *A* or a gap ('-'), the second element is always from sequence *B* or a gap and the third element is always from sequence *C* or a gap.

Definition 3. *A null triple is consisted of three gaps.*

In here, we only consider the alignments without null triple. Therefore, there are seven types for non-null aligned triples according to the number and the position of appearances of gaps ('-'). An example of the result for 3*SA* is illustrated in Figure 1. Figure 1 shows these three sequences *A*, *B* and *C*, and the result of aligning them. Each column in the alignment is an aligned triple.

Definition 4. *A block in an alignment is a contiguous subsequence of aligned triples of the same type bounded by aligned triples of other types or the end of the alignment.*

Definition 5 *An i-gap block is a block consisting of triples in which gap appears exactly i times.*

There are only 1-gap blocks and 2-gap blocks in an alignment of three sequences as shown in Figure 1 since we only consider the alignments without null triple here. For example, in Figure 1, an aligned triple (-, *S*, -) is a 2-gap block and another triple (H, -, H) is a

1-gap block. Given a scoring function f: $\Sigma\times\Sigma\rightarrow$R that assigns a value to each combination of two elements in alphabet Σ of a triple and a gap penalty function, the score of an alignment is the sum of the values of each aligned triple assigned by f minus gap penalties. For example, it can be simply defined that function f as $f(a,a) = 10$ for each a in Σ or $f(a, b) = -20$ for all a, b in Σ with $a\neq b$. For protein sequences, the function f is usually defined through a scoring matrix such as PAM250 or BLOSUM62.

For gap penalties, an affine gap penalty function is commonly used in practice. Let q_1 be a gap-open penalty for each 1-gap block and let r_1 be a gap-extension penalty for each triple of 1-gap block, where q_1 and r_1 are two non-negative numbers. The affine gap penalty function takes p_1 for a triple of 1-gap block, where p_1 is defined as:

$$p_1 = \begin{cases} q_1 + r_1, \text{if this triple is the start of the block.} \\ r_1, \text{otherwise.} \end{cases}.$$

We also define q_2, r_2 and p_2 for 2-gap block in the same way. Let x be the score of a triple. For any a, b and c in Σ, we have a score x as:

$$x(a,b,c) = f(a,b) + f(a,c) + f(b,c)$$
$$x(a,b,-) = x(a,-,b) = x(-,a,b) = f(a,b) - p_1 .$$
$$x(a,-,-) = x(-,a,-) = x(-,-,a) = -p_2$$

The score of an alignment is the sum of the score x of each triple. For example, the score of the alignment for sequences A, B and C shown in Figure 1 is 246 if $f(a,a) = 10$ for each a in Σ, $f(a, b) = -20$ for all a, b in Σ with $a\neq b$, $q_1 = q_2 = 12$ and $r_1 = r_2 = 2$. The goal of 3SA problem is to find an alignment with best score, which is called an optimal alignment.

3.2. 3*SA* algorithm

Once a scoring mechanism is given, the optimal alignment of three sequences can be found by using the dynamic programming approach. Let $S(m, n, l)$ be the score of an optimal alignment of three sequences $A_{1,m}$, $B_{1,n}$ and $C_{1,l}$ with lengths m, n and l, respectively. The score of $S(i, j, k)$ can be computed along with auxiliary matrices according to the recurrences [8]. In the recurrences, the matrices E, F and G save the scores that a gap opens at position j of sequence B, position i of sequence A, and position k of sequence C, respectively. The matrices H, I and J save the scores that position i of sequence A, position j of sequence B and position k of sequence C match two-gaps, respectively. Once $S(m, n, l)$ is computed, an optimal alignment of best score $S(m, n, l)$ can be found by a trace back procedure. Since the entire matrices have to

$$S(i,j,k) = \begin{cases} 0 & \text{if } i=0, j=0 \text{ and } k=0 \\ -(q_2 + r_2 * i) & \text{if } i>0, j=0 \text{ and } k=0 \\ -(q_2 + r_2 * j) & \text{if } i=0, j>0 \text{ and } k=0 \\ -(q_2 + r_2 * k) & \text{if } i=0, j=0 \text{ and } k>0 \\ \max\{G(i,j,k), H(i,j,k), I(i,j,k)\} & \text{if } i>0, j>0 \text{ and } k=0 \\ \max\{E(i,j,k), H(i,j,k), J(i,j,k)\} & \text{if } i>0, j=0 \text{ and } k>0 \\ \max\{F(i,j,k), I(i,j,k), J(i,j,k)\} & \text{if } i=0, j>0 \text{ and } k>0 \\ \max\begin{Bmatrix} E(i,j,k), F(i,j,k), G(i,j,k), \\ H(i,j,k), I(i,j,k), J(i,j,k), \\ S(i-1,j-1,k-1) + x(a_i, b_j, c_k) \end{Bmatrix} & \text{if } i>0, j>0 \text{ and } k>0 \end{cases}$$

$$E(i,j,k) = \begin{cases} S(i,j,k) - q_1 & \text{if } i=0 \text{ or } k=0 \\ \max\{E(i-1,j,k-1), S(i-1,j,k-1) - q_1\} + f(a_i, c_k) - r_1 & \text{if i}>0 \text{ and } k>0 \end{cases}$$

$$F(i,j,k) = \begin{cases} S(i,j,k) - q_1 & \text{if } j=0 \text{ or } k=0 \\ \max\{F(i,j-1,k-1), S(i,j-1,k-1) - q_1\} + f(b_j, c_k) - r_1 & \text{if } j>0 \text{ and } k>0 \end{cases}$$

$$G(i,j,k) = \begin{cases} S(i,j,k) - q_1 & \text{if } i=0 \text{ or } j=0 \\ \max\{G(i-1,j-1,k), S(i-1,j-1,k) - q_1\} + f(a_i, b_j) - r_1 & \text{if i}>0 \text{ and } j>0 \end{cases}$$

$$H(i,j,k) = \begin{cases} S(i,j,k) - q2 & \text{if } i=0 \\ \max\{H(i-1,j,k), S(i-1,j,k) - q_2\} - r_2 & \text{if } i>0 \end{cases}$$

$$I(i,j,k) = \begin{cases} S(i,j,k) - q2 & \text{if } j=0 \\ \max\{I(i,j-1,k), S(i,j-1,k) - q_2\} - r_2 & \text{if } j>0 \end{cases}$$

$$J(i,j,k) = \begin{cases} S(i,j,k) - q2 & \text{if } k=0 \\ \max\{J(i,j,k-1), S(i,j,k-1) - q_2\} - r_2 & \text{if } k>0 \end{cases}$$

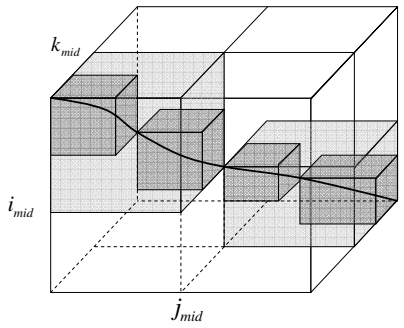


Figure 2. Schematic diagram of divide-and-conquer approach.

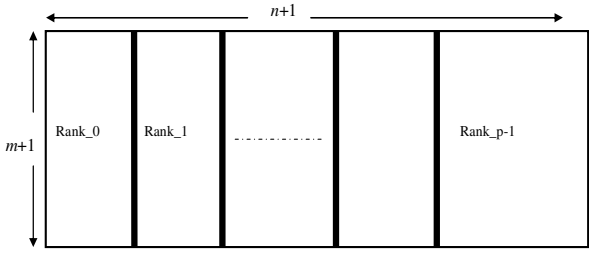


Figure 3. The concept of partitioning a 2D matrix by using Block partition scheme.

be kept, the trace back procedure requires O(*mnl*) space. Note that it only takes four two-dimensional (2*D*) matrices and a few one-dimensional (1*D*) arrays if only the best score *S*(*m, n, l*) is needed. For reducing the space requirement to O(*mn*), assumed that *l* is larger than *m* and *n*, the divide-and-conquer technique of Hirschberg [10] is applied. The central idea is to determine a middle point (i_{mid}, j_{mid}, k_{mid}), which is a point on an optimal path from *S*(0, 0, 0) to *S*(*m, n, l*).

After determining a middle point (i_{mid}, j_{mid}, k_{mid}), the original 3*SA* problem can be divided into two smaller 3*SA* problems. Then these smaller 3*SA* problems are divided recursively. Finally, an optimal 3*SA* is obtained by merging the series of the computed middle points. Figure 2 illustrates this idea.

For details, recall that *S*(*m, n, l*) is the score of an optimal alignment of three sequences $A_{1,m}$, $B_{1,n}$ and $C_{1,l}$, the matrices introduced above are computed in increasing order of indices. Another set of matrices can be defined symmetrically. Let *R, U, V, W, X, Y,* and *Z* be the set of matrices which are computed in decreasing order of indices, where *R* corresponds to *S*, *U* corresponds to *E*, *V* corresponds to *F*, *W* corresponds to *G*, *X* corresponds to *H*, *Y* corresponds to *I*, Z corresponds to *J*. Therefore, *R*(0, 0, 0), equal to *S*(*m, n, l*), is the score of an optimal alignment of three sequences $A_{m,1}$, $B_{n,1}$ and $C_{l,1}$. To find the middle point (i_{mid}, j_{mid}, k_{mid}), first set k_{mid} to $\lfloor l/2 \rfloor$. In the forward phase, compute the matrices *S* through *J* for $0 \le i \le m$, $0 \le j \le n$ and $0 \le k \le k_{mid}$, and save four 2*D* matrices *S*(*i, j, k*), *E*(*i, j, k*), *F*(*i, j, k*) and *J*(*i, j, k*). Note that, k_{mid} is a character in sequence *C*, we do not have to consider the cases in which k_{mid} is a gap. In the reverse phase, compute the matrices *R* through *Z* for $0 \le i \le m$, $0 \le j \le n$ and $k_{mid} \le k \le l$, and save four 2*D* matrices *R*(*i, j, k*), *U*(*i, j, k*), *V*(*i, j, k*) and *Z*(*i, j, k*). The middle point (i_{mid}, j_{mid}, k_{mid}) of an optimal alignment of three sequences *A*, *B* and *C* is one which has the score:

$$\max\begin{Bmatrix} S(i,j,k_{mid})+R(i,j,k_{mid}), \\ E(i,j,k_{mid})+U(i,j,k_{mid})+q_1, \\ F(i,j,k_{mid})+V(i,j,k_{mid})+q_1, \\ J(i,j,k_{mid})+Z(i,j,k_{mid})+q_2 \end{Bmatrix} \text{ for } 0 \le i \le m \text{ and } 0 \le j \le n$$

Note that we need to add a gap-open penalty q_1 or q_2 to some pairs of matrix because an extra q_1 or q_2 is charged when two gaps of the same type merged into a single gap. Here, only some 2*D* matrices and some 1*D* arrays are used to find the middle point (i_{mid}, j_{mid}, k_{mid}). Once the middle point (i_{mid}, j_{mid}, k_{mid}) is found, we recursively compute an optimal alignment of subsequences $A_{1,}$ i_{mid}, $B_{1,}$ j_{mid} and $C_{1,}$ k_{mid} and another one of subsequences A_{m}, $i_{mid+1,}$ $B_{n,}$ j_{mid+1} and C_{l}, $k_{mid+1,}$ respectively. The optimal 3*SA* is obtained by merging the series of the computed middle points. It is obvious that the space required by this algorithm with a divide-and-conquer technique is O(*mn*). Here, we briefly prove the time complexity of this algorithm retains O(*mnl*). Let *T* be the time required to compute the middle point (i_{mid}, j_{mid}, k_{mid}), *T* will be *cmnl*, where *c* is a constant coefficient. Once the middle point is found, the original problem *T*(*m*, *n*, *l*) is divided into two subproblems *T*(*m*, *n*, $\lfloor l/2 \rfloor$) and *T*(*m*, *n*, $\lceil l/2 \rceil$). The time required to compute the middle points of the two subproblems is *T*/2. We can see that the time required to compute the middle points of subproblems is half the time of original problems. The total time required to compute the optimal alignment is $T(m, n, l) = T + T(m, n, \lfloor l/2 \rfloor) + T(m, n, \lceil l/2 \rceil) \le 2T$ and consequently the time complexity of this algorithm is still O(*mnl*).

3.3. *P3SA* algorithm

In this section, *P*3*SA* algorithm is proposed. The critical cost of 3*SA* algorithm is the part of computation of 2*D* matrices, such as matrices *S*, *E*, *F* and *J*. Hence, it is worthy to reduce computational time and space requirement for this part. The *P*3*SA* algorithm will partition each of 2*D* matrices into *p* parts by using Block partition scheme first. Then, each processor will execute 3*SA* algorithm with corresponding matrices. Finally, the result of three sequences alignment will be given by merging partial results from each processor. The *P*3*SA* algorithm can be divided into four parts, *initiation and allocation phase*, *forward phase*, *reverse phase* and *determine a middle point phase*. In the following, each phase will be illustrated, respectively.

A. *initiation and allocation phase*

After determining an initial middle point, the original 3*SA* problem can be divided into two smaller problems, called forward phase and reverse phase. In the forward phase, four 2*D* matrices *S, E, F, J* and other 1*D* array *G*, *H*, *I* need to be computed. In the reverse phase, four 2*D* matrices *R, U, V*, *Z* and other 1*D* array *W*, *X*, *Y* need to be computed. Therefore, in this phase, eight 2*D* matrices *S, E, F, J*, *R, U, V* and *Z* will be partitioned into *p* parts. Figure 3 shows the concept of partitioning a 2*D* matrix by using Block partition scheme.

Each processor only needs to compute the corresponding part of matrices in the forward and reverse phases and then determine the candidate middle point according to their partial results. The real middle point will be determined by merging all candidate

middle points. The steps of this phase are shown below:

***Initiation and allocation*:**
Step0: *Initialize p processors from rank_*0 *to rank_p*-1.
Step1: *Input sequences A, B and C with lengths m, n and k, respectively.*
Step2: *set* k_{mid} *is* $\lfloor l/2 \rfloor$ as an initial middle point.
Step3: *Allocate* eight *2D matrices and other 1D arrays for each processor rank_*0 *to rank_p*-1.

B. *Forward and Reverse phases*

In these two phases, each processor only needs to compute the partial 2*D* matrices and other corresponding 1*D* arrays. Due to the dependency of dynamic programming, each processor can compute the partial matrix after receive the necessary data from its neighbor. It will be a pipelining technique. The steps of these two phases are shown below, respectively:

***Forward phase*:**
For k=0 *to* $k=k_{mid}$ **do**
Processor *rank_*0:
Step0: *Compute all partial 2D matrices and 1D arrays.*
Step1: *Send the last columns of each 2D matrices to rank_*1.
Processor *rank_*(*i*), for $1 \le i \le p-2$ **:**
Step0: *Receive the last columns of each 2D matrices from rank_*(*i*-1).
Step1: *Compute all partial 2D matrices and 1D arrays.*
Step2: *Send the last columns of each 2D matrices to rank_*(*i*+1).
Processor *rank_p*-1:
Step0: *Receive the last columns of each 2D matrices from rank_p*-2.
Step1: *Compute all partial 2D matrices and 1D arrays.*
End for

***Reverse phase*:**
For $k=l$ *to* $k=k_{mid}$ **do**
Processor *rank_p*-1:
Step0: *Compute all partial 2D matrices and 1D arrays.*
Step1: *Send the last columns of each 2D matrices to rank_p*-2.
Processor *rank_*(*i*), for $1 \le i \le p-2$ **:**
Step0: *Receive the last columns of each 2D matrices from rank_*(*i*+1).
Step1: *Compute all partial 2D matrices and 1D arrays.*
Step2: *Send the last columns of each 2D matrices to rank_*(*i*-1).
Processor *rank*_0:
Step0: *Receive the last columns of each 2D matrices from rank_*1.
Step1: *Compute all partial 2D matrices and 1D arrays.*
End for

C. *Determine a middle point phase*

After the forward and reverse phases, a new middle point can be determined from all processors. Each processor can find a candidate middle point with best score by merging its results of forward and reverse phases first. Then, each processor send its candidate middle point to processor rank_0. Finally, processor rank_0 will determine a new middle point by comparing the scores of all candidates to find a real best one. The steps of this phase are shown below.

Determine a middle point
For processor *Rank_*1 to *Rank_p*-1
Step0: *Find a middle point with best score by merging its results of forward and reverse phases.*
Step1: *Send this middle point to Rank_*0.
For processor *Rank_0:*
Step0: *Find a middle point with best score by merging its results of forward and reverse phases.*
Step1: *Receive middle points from other processors as candidate middle points.*
Step2: *Determine the real middle point by comparing these candidates*.

When a new middle point is determined, each of subproblems can be divided into two smaller problems. The *P*3*SA* algorithm will execute these four phases recursively to find all of middle points as an optimal path from *S*(0, 0, 0) to *S*(*m, n, l*). The optimal alignment of three sequences is given by this path.

4. Analysis

In this section, the time and the space complexities of *P*3*SA* method will be proved. The theoretical performance of *P*3*SA* method by considering computation and communication time will be also analyzed. From Figure 3, we can see that, in a $(m+1)\times(n+1)$ matrix, each of p processors handles $\lfloor n+1/p \rfloor$

columns except processor rank_*p*-1. Therefore, each processor takes O(*mn/p*) time complexity and requires O(*mn/p*) space complexity for each (*m*+1)*(*n*+1) matrices. When the dimension *k* increases from 1 to *l*, each processor takes O(*mn/p*) time complexity. However, each processor only requires O(*mn/p*) space complexity by reusing the space for *k* = *i* to update the values for *k*= *i*+1 with an addition one-dimensional array as used in the Hirschberg's technique. In practice, each processor will compute eight 2*D* matrices and other 1*D* arrays and the dimension *k* will be divided into two parts recursively. Hence, *P*3*SA* method takes O(*mn/p*) time complexity and O(*mn/p*) space complexity.

The details of theoretical performance for *P*3*SA* method are analyzed below. Some parameters are introduced here. Let t_1 be the computation time of one array element; let t_2 be a startup time between two processors with a communication; let t_3 be the transmission time of one array element in a communication channel. In order to simplify the analysis, a (*m*+1)×(*n*+1) matrix will be regarded as a *m*×*n* matrix and *m* and *n* can be divided with *p*. As mentioned in Section 3.3, in the forward phase (or reverse phase), processor rank_1 (or rank_*p*-2) can compute each two-dimensional matrix when it receive the data from processor rank_0 (or rank_*p*-1). There is an idle time between processor rank_0 and rank_1 and the time is equivalent to the time of computing a matrix with size of *mn/p* by processor rank_0 adds the communication time of sending a column with size of *m* from processor rank_0 to processor rank_1. The computing time is $(mn/p)t_1$ and the communication time is (t_2+t_3m). Similarly, processor rank_2 can compute the matrix when processor rank_1 computed its matrix and then send the last column of this matrix to processor rank_2. Hence, last processor rank_*p*-1 starts to compute its matrix when it is waiting for processor rank_*p*-2 completes its part and then send the column to processor rank_*p*-1. The idle time of last processor is $((p-1)mn/p)t_1 + (t_2 + t_3m)(p-1)$. The last processor rank_*p*-1 will complete its matrix with the computing time $(mn/p)t_1$. In an ideal environment, when last processor rank_*p*-1 completed its part, the processor rank_*p*-2 also complete its next plane (dimension *k* increases). The last processor rank_*p*-1 can start to compute its next plane when it is waiting for processor rank_*p*-2 send the column to it (as a pipeline). Therefore, the required time of last processor rank_*p*-2 for finishing forward phase is the computing time $(mn/p)\times(l/2)t_1$ adds the communication time $(t_2 + t_3m)((l/2)-1)$ after the idle time. Figure 4 illustrates this theoretical analysis concept.

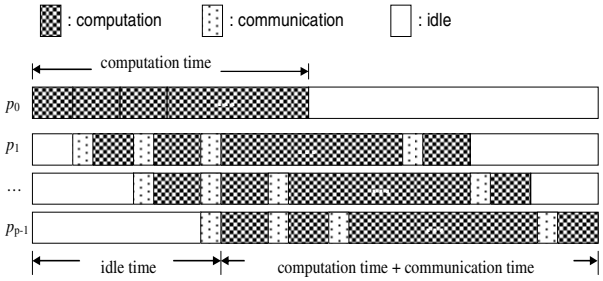


Figure 4. The theoretical analysis concept.

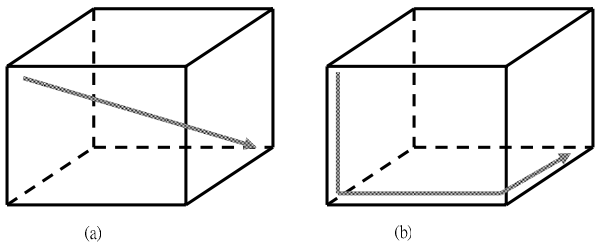


Figure 5. A random data set with complete match (a) and complete mismatch (b).

The required time of last processor rank_0 for finishing reverse phase is equal to that of forward phase. After each processor completed the forward and reverse phases, each processor needs to add these two results to find a candidate middle point and then send it to the processor rank_0. After processor rank_0 received all candidate middle points, it will determine a new middle point. The required time of processor rank_0 for determining a new middle point is $(mn/p)t_1 + (t_2 + t_3)(p-1)$. After the initial middle point (k_{mid} *is* $\lfloor l/2 \rfloor$), the total time for finding a new middle point is $t_1(mn/p)(l+2p-1) + t_2(l+3p-5) + t_3[m(l+2p-4)+p-1]$. The *P*3*SA* algorithm will execute these four phases recursively to find all of middle points for each matrix. The time related to t_1 and t_3 will be double. The time related to t_2 is direct proportional to the number of communications occur and it will be multiplied by a variable *L*, where *L* is a parameter proportional to the length of the optimal path, $\max(m,n,l) \le L \le m+n+l$.

The total time required by *P*3*SA* method to find all of middle points for each matrix is $\{t_1(mn/p)(l+2p-1) + t_3[n(l+2p-4)+p-1]\}\times 2 + [t_2(l+3p-5)]*L$. The time required by sequential 3*SA* method to find all of middle points for each matrix

is $2(mnl)t_1$. For sequential 3*SA* and *P*3*SA* methods, they both need to compute four 2*D* matrices. The costof computing other 1*D* arrays is omitted here since it is very small if comparing it with the cost of computing 2*D* matrices.

The speed-up ratio of *P*3*SA* method is $\frac{2(mnl)t_1}{\{t_1(\frac{mn}{p}(l+2p-1)+t_3[n(l+2p-4)+p-1]\}*2+[t_2(l+3p-5)]*L}$.

5. Experimental Results

The *P*3*SA* method has been implemented by MPI + C code, and tested on the NCHC Formosa Linux Cluster with a clock rate of 2.8G Hz, 2GB memory and 1000Mbps switch. A random data set with complete match and mismatch cases as those used in [20], shown in Figure 5, are used to evaluate *P*3*SA* method. The runtime of *P*3*SA* method with various numbers of processors and various lengths of input three sequences are shown in Table 1. From Table 1, (1) we can see that the 3*SA* method is not applicable when the sequence length is larger than 8k, however, it will be applicable for *P*3*SA* method. (2) The runtime of complete match and complete mismatch cases both can be reduced when the number of processors increases. It shows that *P*3*SA* method is useful for the optimal alignment of three sequences. Figure 6 shows the speed-up ratios of complete match and complete mismatch cases with various numbers of processors and various lengths of input three sequences. From Figure 6, (1) we can see that the *P*3*SA* method obtains a satisfied speed-up ratio for complete match and complete mismatch cases. (2) The speed-up ratio will increase when the sequence length increases.

6. Conclusions

In this paper, *P*3*SA* method is proposed to solve the 3*SA* problem. The *P*3*SA* method requires $O(n^2/p)$ space complexity and $O(n^3/p)$ time complexity. Both theoretical analysis and experimental tests have been presented. The experimental results show that a good performance and a satisfied speed-up are achieved by using *P*3*SA* method. In the future, we plan to apply *P*3*SA* method to the following applications. (1) Testing the performance of *MSA* by using *P*3*SA* method as a basic step instead of 2*SA* method. It will help us to find the conditions used to chose the 3SA or 2SA for practical applications. (2) Finding important patterns or residues on DNA/protein sequences by using *P*3*SA* method.

Acknowledgments

Post doctor fellowship of Chun Yuan Lin is supported by NSC under contract NSC94-2627-B-007-002 and NSC 95-2221-E-007-031. The authors would like to thank anonymous referees for their comments. These comments are useful for this paper and the research work in the future.

References

[1]. L. Allison, "A Fast Algorithm for the Optimal Alignment of Three Strings," *J. Theor. Biol.* Vol. 164, 1993, pp. 261-269.

[2]. S. Aluru, N. Futamura and K. Mehrotra, "Parallel biological sequence comparison using prefix computations," *J. Parallel and Distributed Computing*, Vol. 63, Issue 3, 2003, pp. 264-272.

[3]. P. Bonizzoni and G. D. Vedova, "The complexity of multiple sequence alignment with SP-score that is a metric," *Theoretical Computer Science*, Vol. 259, Issue1–2, 2001, pp. 63-79.

[4]. H. Carrillo and D. Lipman, "The multiple sequence alignment problem in biology," *SIAM J. Applied Math.*, Vol. 48, 1988, pp. 1073-1082.

[5]. S.C. Chan, A.K.C. Wong and D.K.Y. Chiu, "A survey of multiple sequence comparison methods," *Bulletin of Mathematical Biology,* Vol. 54, 1992, pp. 563-598.

[6]. N. Futamura, S. Aluru, and X. Huang, "Parallel Syntenic Alignment," *Proc. Int'l Conf. High Performance Computing*, 2002, pp. 420-430.

[7]. D.Gusfield, *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*, Cambridege University Press, NY., 1997.

[8]. O. Gotoh, "An improved algorithm for matching biological sequences," *J. Molecular Biology*, Vol. 162, 1982, pp. 705-708.

[9]. O. Gotoh, "Alignment of three biological sequences with an efficient traceback," *J. Theor. Biol.* Vol. 121, 1986, pp. 327-337.

[10]. D.S. Hirschberg, "A Linear Space Algorithm for Computing Maximal Common Subsequences," *Comm. ACM*, Vol. 18, No.6, 1975, pp. 341-343.

[11]. X. Huang, "A Space-Efficient Parallel Sequence Comparison Algorithm for a Message-Passing Multiprocessors," *Int'l j. Parallel Programming*, Vol. 18, No. 3, 1989, pp. 223-239.

[12]. X. Huang, "A Space-Efficient Algorithm for Local Similarities," *Computer Applications in the Biosciences*, Vol. 6, No. 4, 1990, pp.373-381.

[13]. X. Huang, "Alignment of Three Sequences in Quadratic Space," *ACM SIGAPP Applied Computing,* Vol. 1, Issue 2, 1993, pp. 7-11.

[14]. M.S. Johnson and R.F. Doalittle, "A Method for the Simultaneous Alignment of Three or More Amino Acid Sequences," *J Mol Evol.* Vol. 23, 1986, pp. 267-78.

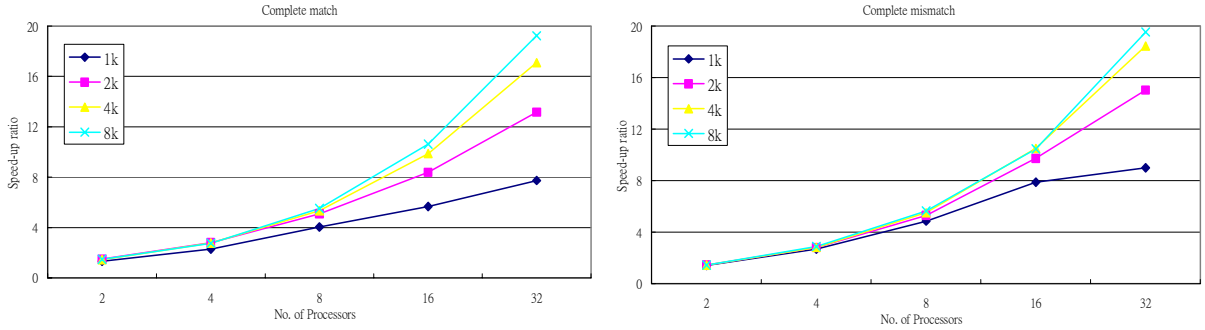
[15]. E.W. Mayers and W. Miller, "Optimal Alignments in Linear Space," *Computer Applications in the*

Biosciences, Vol. 4, No. 1, 1988, pp. 11-17.
[16]. M. Murata, J.S. Richardson, and J.L. Sussman, "Simultaneous Comparison of Three Protein Sequences," *Proc. Natl. Acad. Sci USA*, Vol. 82, 1985, pp. 3073-3077.
[17]. H.B. Nicholas, A.J. Ropelewski and D.W. Deerfield, "Strategies for Multiple Sequence Alignment," *Biotechniques,* Vol. 32, 2002, pp. 592-603.
[18]. S.E. Needleman and C.D. Wunsch, "A general method applicable to the search for similarities in the amino-acid sequence of two proteins," *J. Molecular Biol.*, Vol. 48, 1970, pp. 443-453.
[19]. C. Notredame, "Recent progresses in multiple sequence alignment: a survey," *Pharmacogenomics,* Vol. 3, 2002, pp. 131-144.
[20]. S. Rajko and S. Aluru, "Space and Time Optimal Parallel Sequence Alignments," *IEEE Trans. Parallel and Distributed Systems*, Vol. 15, Issue 12, 2004, pp. 1070-1081.
[21]. D. Sankoff, "Simultaneous Solution of the RNA Folding, Alignment and Protosequence Problems," *SIAM J. Appl. Math.*, Vol. 45, Issue 5, 1985, pp. 810-825.
[22]. J. Setubal and J. Meidanis, *Introduction to Computational Molecular Biology.* PWS Publishing Company, 1997.
[23]. T.F. Smith and M.S. Waterman, "Identification of Common Molecular Subsequences," *J. Molecular Biology*, vol. 147, 1981, pp.195-197.
[24]. C.Y. Tang, C.L. Lu, M.D.T. Chang, Y.T. Tsai, Y.J. Sun, K.M. Chao, J.M. Chang, Y.H. Chiou, C.M. Wu, H.T. Chang and W.I. Chou, "Constrained Multiple Sequence Alignment Tool Development and Its Application to RNase Family Alignment," *Journal of Bioinformatics and Computational Biology*, Vol. 1, 2003, pp. 267-287.
[25]. L. Wang and T. Jiang, "On the complexity of multiple sequence alignment," *J. Computational Biology*, Vol. 1, No. 4, 1994, pp. 337–348.

Table 1. Runtime of *P3SA* method tested on Formosa Linux Cluster.

No. of processors / Input Size	1	2	4	8	16	32
Complete match						
1k×1k×1k	85	64	37	21	15	11
2k×2k×2k	737	495	263	145	88	56
4k×4k×4k	5745	3965	2079	1076	582	336
8k×8k×8k	47620	32373	17340	8634	4484	2476
Complete mismatch						
1k×1k×1k	126	90	47	26	16	14
2k×2k×2k	1022	714	366	193	105	68
4k×4k×4k	8161	5673	2906	1482	777	442
8k×8k×8k	66828	46383	23174	11849	6388	3418

Time: second.



(a) Complete match (b) Complete mismatch

Figure 6. Speed-up ratios of *P3SA* method tested on Formosa Linux Cluster.