A Parallel LCS Algorithm for Biosequences Alignment

Wei Liu
Institute of Information Science and Technology, Nanjing University of Aeronautics and Astronautics
Nanjing 210093, China
+8613951735560
yzliuwei@126.com

Ling Chen
Department of Computer Science ,Yangzhou University
Yangzhou 225009 China
+865147899311
chen@yzcn.net

Lingjun Zou
Department of Computer Science ,Yangzhou University
Yangzhou 225009 China
+865147993908
njzoulingjun@126.com

ABSTRACT
Searching for the longest common substring (LCS) of biosequences is one of the most important tasks in Bioinformatics. A fast algorithm for LCS problem named FAST_LCS is presented. The algorithm first seeks the successors of the initial identical character pairs according to a successor table to obtain all the identical pairs and their levels. Then by tracing back from the identical character pair at the largest level, the result of LCS can be obtained. For two sequences X and Y with lengths n and m, the memory required for FAST_LCS is $\max\{8*(n+1)+8*(m+1),L\}$, here L is the number of identical character pairs and time complexity of parallel implementation is $O(|\mathrm{LCS}(X,Y)|)$, here, $|\mathrm{LCS}(X,Y)|$ is the length of the LCS of X,Y. Experimental result on the gene sequences of *tigr* database shows that our algorithm can get exactly correct result and is faster and more efficient than other LCS algorithms.

Categories and Subject Descriptors
J.3 [**Life and Medical Science**] Biology and genetics

General Terms
Algorithms

Keywords
Bioinformatics, Parallel processing, Biosequences alignment

1. INTRODUCTION
Biological sequence [1] can be represented as a sequence of symbols. For instance, a protein[2] is a sequence of 20 different letters (amino acids), and DNA sequences (genes) can be represented as sequences of four letters A,C,G and T, corresponding to the four sub-molecules forming DNA. When biologists find a new sequence, they want to know what other sequences it is most similar to. Sequence comparison[3-5] has been used successfully to establish the link between cancer-causing genes and a gene evolved in normal growth and development. One way of detecting the similarity of two or more sequences is to find their longest common subsequence (LCS).

The longest common subsequence problem is to find a substring that is common to two or more given strings and is the longest one of such strings. Presented in 1981， Smith-Waterman algorithm [6] was a well known LCS algorithm which was evolved by the Needleman-Wunsch [7] algorithm, and can guarantee the correct result. Aho et al.[8] gave a lower bound of $O(mn)$ on time for the LCS problem using a decision tree model. It is shown in [9] that the problem can be solved in $O(mn)$ time using $O(mn)$ space by dynamic programming. Mayers and Miller[10] use the skill proposed by Hirschberg [11] to reduce the space complexity to $O(m+n)$ on the premise of the same time complexity. To further reduce the computation time, some parallel algorithms[12-14] have been proposed for the LCS problem on different computational models. On CREW-PRAM model, Aggarwal [15] and Apostolico et al [16] independently proposed an $O(\log m \log n)$ time algorithm using $mn/\log m$ processors. Lu et al [17] designed two parallel LCS algorithms, one used $mn/\log m$ processors with a time complexity of $O(\log^2 n+\log m)$, and the other used $mn/(\log^2 m \operatorname{loglog} m)$ processors with a running time of $O(\log^2 m \operatorname{loglog} m)$. On the CRCW-PRAM model, Apostolico et al [16] gave an $O(\log n (\operatorname{loglog} m)^2)$ time algorithm using $mn/\mathrm{loglogm}$ processors. Babu and Saxena [18] improved these algorithms on the CRCW-PRAM model. They designed an $O(\log m)$ algorithm with mn processors and an $O(\log^2 n)$ time optimal parallel algorithm. Many parallel LCS algorithms have also been proposed on systolic arrays. Robert et al [19] proposed a parallel algorithm with $n+5m$ steps using $m(m+1)$ processing elements. Chang et al [20] put forward an algorithm with $4n+2m$ steps using mn processing elements. Luce et al [21] designed a systolic array with $m(m+1)/2$ processing elements and $n+3m+q$ steps where q is the length of the LCS. Freschi and Bogliolo [22] addressed the problem of computing the LCS between run-length-encoded (RLE) strings. Their algorithm requirs $O(m+n)$ steps on a systolic array of $M+N$ processing elements, where M and N are the lengths of the original strings and m and n are the number of runs in their RLE representation.

In this paper, we present a fast algorithm named FAST_LCS for LCS problem. The algorithm first seeks the successors of the initial identical character pairs according to a successor table to obtain all the identical pairs and their levels. Then by tracing back from the identical character pair at the largest level, the result of LCS can be obtained. For two sequences X and Y with lengths n

INFOSCALE 2007, June 6-8, Suzhou, China

DOI 10.4108/infoscale.2007.954

and m, the memory required for FAST_LCS is $\max\{8*(n+1)+8*(m+1), L\}$, here L is the number of identical character pairs and time complexity of parallel implementation is $O(|\mathrm{LCS}(X,Y)|)$, here, $|\mathrm{LCS}(X,Y)|$ is the length of the LCS of X,Y. Experimental result on the gene sequences of *tigr* database shows that our algorithm can get exactly correct result and is faster and more efficient than other LCS algorithms.

2. THE IDENTICAL PARE AND ITS SUCCESSOR TABLE

Let $X=(x_1, x_2, \dots, x_n)$, $Y=(y_1, y_2, \dots, y_m)$ be two biosequences, where $x_i, y_i \in \{A,C,G,T\}$. We can define an array CH of the four characters so that CH(1)="*A*", CH(2)="C", CH(3)="G" and CH(4)="T". To find their longest common subsequence, we first build the successor tables of the identical characters for the two strings. The successor tables of X and Y are denoted as TX and TY which are $4*(n+1)$ and $4*(m+1)$ two dimensional arrays. $TX(i, j)$ is defined as follows.

Definition1. For the sequence $X=(x_1, x_2, \dots, x_n)$, its successor table TX of identical character is defined as :

$$TX(i,j)=\begin{cases}\min\{k \mid k \in SX(i,j)\} & SX(i,j)\neq\phi \\ - & \text{otherwise}\end{cases} \quad (1)$$

Here, $SX(i, j)=\{k \mid x_k=\mathrm{CH}(i), k>j\}$, $i = 1,2,3,4$, $j = 0,1,\dots n$. It can be seen from the definition that if $TX(i, j)$ is not "$-$", it indicates the position of the next character identical to CH(i) after the jth position in sequence X. If $TX(i, j)$ is equal to "$-$", it means there is no character CH(i) after the jth position .

Example 1 Let X = "T G C A T A", Y = "A T C T G A T". Their successor tables TX and TY are*:*

TX:

i	CH(i)	0 1 2 3 4 5 6
1	**A**	4 4 4 4 6 6 -
2	**C**	3 3 3 - - - -
3	**G**	2 2 - - - - -
4	**T**	1 5 5 5 5 - -

TY:

i	CH(i)	0 1 2 3 4 5 6 7
1	**A**	1 6 6 6 6 6 - -
2	**C**	3 3 3 - - - - -
3	**G**	5 5 5 5 5 - - -
4	**T**	2 2 4 4 7 7 7 -

Definition2. For the sequences X and Y, if $x_i = y_j$, we call them an identical character pair of X and Y, and denote it as (i, j). The set of all the identical character pairs of X and Y is denoted as $S(X, Y)$.

Definition3. Let (i, j) and (k, l) be two identical character pairs of X and Y . If $i<k$ and $j<l$, we call (i, j) a predecessor of (k, l), or (k, l) a successor of (i, j), and denote them as $(i, j)\prec(k, l)$.

Definition4. Let $P(i, j) = \{(r, s) \mid (i, j) \prec (r, s), (r, s) \in S(X,Y)\}$ be the set of all the successors of identical pair (i, j), if $(k, l) \in P(i, j)$ and there is no $(k', l') \in P(i, j)$ satisfying the condition: $(k', l') \prec (k, l)$, we call (k, l) the direct successor of (i, j), and denoted it as $(i, j) \prec (k, l)$.

Definition5. If an identical pair $(i, j) \in S(X,Y)$ and there is no $(k, l) \in S(X,Y)$ so that $(k, l) \prec (i, j)$, we call (i, j) an initial identical pair.

Definition6. For an identical pair $(i, j) \in S(X,Y)$, its level is defined as follows:

$$level(i,j)=\begin{cases}1 & \text{if } (i,j) \text{ is an initial identical character pair} \\ \max\{level(k,l)+1 \mid (k,l)\prec(i,j)\} & \text{otherwise}\end{cases} \quad (2)$$

From the definitions above, the following lemma can be easily deduced:

Lemma1. Denote the length of the longest common subsequence of X, Y as $|\mathrm{LCS}(X, Y)|$, then $|\mathrm{LCS}(X, Y)|=\max\{\text{level}(i, j) \mid (i, j) \in S(X, Y)\}$.

Proof: Suppose the identical character pairs corresponding to the longest common subsequence of X, Y are (x_{i1}, y_{j1}), (x_{i2}, y_{j2}), … , (x_{ir}, y_{jr}), here $r=|\mathrm{LCS}(X, Y)|$. By Definition5, we know (i_1, j_1) must be an initial identical character pair. By Definition4 and Definition6, we have: $(i_k, j_k) \prec (i_{k+1}, j_{k+1})$, for k=1,2,…,r-1, and the level of (x_{ik}, y_{jk}) is k. Then we can conclude that maximal level of those identical character pairs is r, i.e. $r= \max\{\text{level}(i, j) \mid (i, j) \in S(X, Y)\}$. The reason is as follows: if r is not the maximal level of the identical character pairs of X, Y, there must be an integer $r'>r$ and identical character pairs: $(x_{i1'}, y_{j1'}) \prec (x_{i2'}, y_{j2'}) \prec \dots \prec (x_{ir'}, y_{jr'})$. It corresponds to another common subsequence of X, Y with length $r'>r$. This is in contradiction with the condition $r=|\mathrm{LCS}(X, Y)|$.

Q.E.D.

3. FIGURES/CAPTIONS

To get all the identical pairs and their levels, we first produce the direct successors of the initial pairs in the first time step in parallel using the successor tables. Then the direct successors of all those successors produced in the first step are generated in parallel in the second time step. Repeat these operations of generating the direct successors until no more successors could be produced. Therefore, producing all the direct successors for the identical character pairs is a basic operation in our algorithm.

For an identical character pair $(i, j) \in S(X, Y)$, the operation of producing all its direct successors is as follows:

$$(i,j)\rightarrow\{(TX(k,i),TY(k,j)) \mid k=1,2,3,4, TX(k,i)\neq\text{'–'} \text{ and } TY(k,j)\neq\text{'–'}\} \quad (3)$$

From (3) we can see that this operation is to couple the elements of the ith column of TX and the jth column of TY to get the pairs. For instance, the operation on the identical character pair (2,5) in Example 1 is illustrated as follows:

$$(2\ \ 5)\rightarrow\begin{bmatrix}4 & 6\\ 3 & -\\ - & -\\ 5 & 7\end{bmatrix}\rightarrow\begin{Bmatrix}(4 & 6)\\ (3 & -)\\ (- & -)\\ (5 & 7)\end{Bmatrix}\rightarrow\begin{Bmatrix}(4 & 6)\\ (5 & 7)\end{Bmatrix}$$

Here (3,—) and (—,—) do not represent identical character pairs, they only indicate the end of the process of searching for the

successors in this branch. After discarding (3,－) and (－,－), the successors of (2,5) are just (4,6) and (5,7). It should be point out that the successors produced in the operation are not all direct successors of (*i*, *j*). For example, (5,7) is not the direct successor of (2,5), since (2, 5) ≺ (4, 6) ≺ (5, 7).

Lemma2. For an identical character pair (*i*, *j*), the method illustrated above can produce all its successors.

Proof: By (3), we can produce all direct successors (*TX*(*k*, *i*),*TY*(*k*, *j*)), *k*=1,2,3,4, of (*i*, *j*). According to (1), *TX*(*k*, *i*) is the minimum of *SX*(*i*, *j*). Namely, it is the nearest character identical to CH(*k*) after x_i in string *X*, and *TY*(*k*, *j*) is the nearest character identical to CH(*k*) after y_i in string *Y*. This means that identical pairs (*TX*(*k*, *i*),*TY*(*k*, *j*)), *k*=1,2,3,4, contains all the direct successors of (*i*, *j*) after the operation of producing successors. Consequently, by the same operation on the newly generated identical pairs, we can get all the direct successors of the direct successors of (*i*, *j*). It can be seen that by repeating this operation of producing successors, we can obtain all the successors of (*i*, *j*).

Q.E.D.

It is obvious that (*TX*(*k*, 0),*TY*(*k*, 0)), *k*=1,2,3,4, are all the initial identical pairs of *X* and *Y*. By Lemma 2, we know that starting from those initial identical pairs, all the identical pairs and their levels can be produced. In such process of generating the successors, prune technique can be implemented to remove the identical pairs which can not generate the longest common subsequence so as to reduce the searching space and accelerate the speed of process.

Theorem 1. If two identical character pairs (*i*, *j*) and（*k*, *l*） generated at the same time step satisfy (*k*, *l*)>(*i*, *j*) , then (*k*, *l*) can be pruned without affecting the algorithm to get the longest common subsequence of *X* and *Y*.

Proof: Suppose identical character pairs (*i*, *j*) and (*k*, *l*) are produced by the identical pairs (i_1 , j_1) and (k_1 , l_1) at the same time step. Let the longest common subsequence produced via (k_1, l_1) and (*k*, *l*) be $a_1a_2\ldots a_ma_{m+1}\ldots a_r$, here a_m corresponds to (k_1, l_1) and a_{m+1} corresponds to (*k*, *l*). Similarly, let the subsequence produced via (i_1, j_1) and (*i*, *j*) be $b_1b_2\ldots b_mb_{m+1}\ldots b_s\ldots b_q$, here b_m corresponds to (i_1, j_1) and b_{m+1} corresponds to (*i*, *j*). Since (*k*, *l*)>(*i*, *j*), by Lemma 2, (*k*, *l*) must be produced after (*i*, *j*). Then there must exist b_s ($m+1<s<q$) corresponding to (*k*, *l*). Since $a_ma_{m+1}\ldots a_r$ and $b_sb_{s+1}\ldots b_q$ are both the local longest common subsequences obtained by the operations of producing successors on (*k*, *l*), we have "$a_ma_{m+1}\ldots a_r$" = "$b_sb_{s+1}\ldots b_q$" which means $q-s=r-m$, and $q=r+(s-m)$. Since $s>m$, we have $q>r$. Therefore the subsequence "$a_ma_{m+1}\ldots a_r$", which is produced via (*k*, *l*), can not be included in the longest common subsequence of *X* and *Y*, and (*k*, *l*) can be pruned without affecting the algorithm to get the longest common subsequence of *X* and *Y*.

Q.E.D.

By Theorem 1, the pruning process can be implemented to remove all those redundant identical pairs. At each time step, the algorithm checks all the newly generated identical pairs at the same level to find all such identical pairs (*i*, *j*) and (*k*, *l*) satisfying (*k*, *l*)<(*i*, *j*) and then prune (*i*, *j*).

For instance, (4, 6) and (5, 7) in Example 1 are the successors of the identical pair (2, 5). Since they are generated at the same time step, and (4, 6) ≺ (5, 7), we can prune (5, 7) by Theorem 1.

For another identical character pair (1,1) in Example 1, the operation is illustrated as follows:

$$(1\ 1)\rightarrow\begin{bmatrix}4&6\\3&3\\2&5\\5&2\end{bmatrix}\rightarrow\begin{Bmatrix}(4&6)\\(3&3)\\(2&5)\\(5&2)\end{Bmatrix}\rightarrow\begin{Bmatrix}(3&3)\\(2&5)\\(5&2)\end{Bmatrix}$$

Here four successors of (1, 1) are produced:(4, 6), (3, 3), (2, 5) and (5, 2). Since (3, 3)<(4, 6), by Theorem 1, (4, 6) can be pruned. At the next time step, successors of (3, 3), (2, 5) and (5, 2) are produced by the operation of producing successors. Those successors are checked and pruned as follows:

$$\begin{Bmatrix}(3&3)\\(2&5)\\(5&2)\end{Bmatrix}\rightarrow\left\{\begin{bmatrix}(4&6)\\(5&4)\end{bmatrix}\begin{bmatrix}(4&6)\\(5&7)\end{bmatrix}\begin{bmatrix}(6&6)\end{bmatrix}\right\}\rightarrow\begin{Bmatrix}(4&6)\\(5&4)\end{Bmatrix}$$

Since (6,6)>(5,4), (5,7) > (4,6) and they are generated at the same time step, (6, 6) and (5,7) can be pruned by Theorem 1.

Similar to Theorem 1, other prune operations are also helpful to reduce the searching space. These prune operations are based on the following theorem and corollary.

4. THE OPERATION OF SKIPPING

In the process of generating identical character pairs, we can skip over the subsequences which obviously belong to the longest common subsequence of the given sequences so as to accelerate the process.

Theorem 2 Suppose the subsequence " $x_ix_{i+1}x_{i+2}\ldots x_{i+t}$ " in *X* is identical to the subsequence " $y_jy_{j+1}y_{j+2}\ldots y_{j+t}$ " in *Y*, then the successors of identical pair (*i*,*j*) at level *l* can be replaced by the successors of identical pair (*i*+*t*,*j*+*t*) at level *l*+*t*.

Proof: We need to prove that for all CH(*k*)，*k*=1,2,3,4， the successors of identical pair (*i*+*t*,*j*+*t*) at level *l*+*t* are all successors of identical pair (*i*,*j*) at level *l*. Suppose the character CH(*k*) appears in the subsequence " $x_ix_{i+1}x_{i+2}\ldots x_{i+t}$ " , and (*i*+*s*, *j*+*s*) is a successor of (*i*+*t*,*j*+*t*) identical to CH(*k*), *t*<*s* . Let the characters identical to CH(*k*) in " $x_{i+1}x_{i+2}\ldots x_{i+t}$ " and " $y_{j+1}y_{j+2}\ldots y_{j+t}$ " are $x_{i+k1}x_{i+k2}\ldots x_{i+kq}$, here $1\leqslant k_1\leqslant\ldots\leqslant k_q\leqslant t$. The identical pairs of character CH(*k*) in " $x_{i+1}x_{i+2}\ldots x_{i+t}$ " and " $y_{j+1}y_{j+2}\ldots y_{j+t}$ " are ($i+k_1$,$j+k_1$) ($i+k_2$,$j+k_2$)…($i+k_q$,$j+k_q$). Since (*i*+*s*, *j*+*s*) is a successor of (*i*+*t*,*j*+*t*) identical to CH(*k*) and $k_q\leqslant t<s$, (*i*+*s*, *j*+*s*) must be a successor of ($i+k_q$,$j+k_q$), and is also a successor of (*i*,*j*). If CH(*k*) does not appears in " $x_ix_{i+1}x_{i+2}\ldots x_{i+t}$ " , then the nest character identical to CH(*k*) must appears after (*i*+*t*,*j*+*t*). Since it is a successor of both (*i*,*j*) and (*i*+*t*,*j*+*t*), the successor of (*i*+*t*,*j*+*t*) identical to CH(*k*) is also a successor of (*i*,*j*).

Q.E.D.

Example 2 Let *X* = "TACTTACGACT",*Y*= "GTACTTACCTAG". *X*、*Y* have identical pairs (1,2) identical to "T". After this "T", both *X*、*Y* have identical subsequences "ACTTAC" of length 6. Therefore the successors of (1,2) can be replaced by the successors of identical pair (1+6,2+6)=(7,8), which are (9,11)、(10,9)、(8,12) and (11,10).

To detect such identical subsequences, we define the identical character augment table for the set of sequences.

Definition 7 For the sequence $X=(x_1, x_2, \ldots, x_n)$, its identical character augment table *SX* is a 4*(*n*+1) array defined as :

$$SX(i,j)=\begin{cases} TX(i,j)-j & \text{if } TX(i,j)\neq\text{'–'} \\ \infty & \text{otherwise} \end{cases}$$

It can be seen from the definition that if *SX*(*i*,*j*) is not "∞", it indicates the displacement of the next character identical to CH(*i*) after x_j. From the definition we can see that *SX* can be derived directly from *TX*.

Example 3. For *X*= "TACTTACGACT" in Example 2,

TX is:

i	CH(*i*)	0	1	2	3	4	5	6	7	8	9	10	11
1	A	2	2	6	6	6	6	9	9	9	-	-	-
2	C	3	3	3	7	7	7	7	10	10	10	-	-
3	G	8	8	8	8	8	8	8	8	-	-	-	-
4	T	1	4	4	4	5	11	11	11	11	11	11	-

SX is:

i	CH(*i*)	0	1	2	3	4	5	6	7	8	9	10	11
1	A	2	1	4	3	2	1	3	2	1	∞	∞	∞
2	C	3	2	1	4	3	2	1	3	2	1	∞	∞
3	G	8	7	6	5	4	3	3	1	∞	∞	∞	∞
4	T	1	3	2	1	1	6	5	4	3	2	1	∞

For *Y*= "GTACTTACCTAG" in Example 2, *TY* is:

i	CH(*i*)	0	1	2	3	4	5	6	7	8	9	10	11	12
1	A	3	3	3	7	7	7	7	11	11	11	11	-	-
2	C	4	4	4	4	8	8	8	8	9	-	-	-	-
3	G	1	12	12	12	12	12	12	12	12	12	12	12	-
4	T	2	2	5	5	5	6	10	10	10	10	-	-	-

SY is :

i	CH(*i*)	0	1	2	3	4	5	6	7	8	9	10	11	12
1	A	3	2	1	4	3	2	1	4	3	2	1	∞	∞
2	C	4	3	2	1	4	3	2	1	1	∞	∞	∞	∞
3	G	1	11	10	9	8	7	6	5	4	3	2	1	∞
4	T	2	1	3	2	1	1	4	3	2	1	∞	∞	∞

Lemma 3. If the elements in a column of identical character augment table *SX* are not all equal to ∞, then their must be one element in this column being equal to 1.

Proof: Suppose the elements in the *j*th column of *SX* are not all equal to ∞, then x_j is not the last character of *X*, and their exists x_{j+1} after x_j. Let x_{j+1}=CH(*i*), then *SX*(*i*,*j*)=1。

Q.E.D.

In fact, if the "1" of the *j*th column of *SX* is in the *i*th row, x_{j+1} is equal to CH(*i*). Based on the identical character augment table, we can define the displacement of the identical pairs as follows:

Definition 8. Let (*i*,*j*) be an identical pair of *X*, *Y*, the pairs obtained by grouping the *i*th column of *SX* and the *j*th column of *SY* are called the displacements of the identical pair (*i*,*j*).

The operation of getting the displacements of the identical pair (*i*,*j*) can be defined as follows:

$$(i,j)\rightarrow\{(SX(k,i),SY(k,j))\mid k=1,2,3,4,SX(k,i)\neq\infty \ \ and \ \ SY(k,j)\neq\infty\}$$

The displacements indicate the distances from (*i*,*j*) to the identical pairs after (*i*,*j*). If (*i*,*j*) has a displacement (*r*,*s*), then (*i*+*r*, *j*+*s*) is also an identical pair, namely, $x_{i+r}=y_{j+s}$.

Example 4 The displacements of the identical pair (1,2) in Example 2 are as follows:

$$(1,2)\rightarrow\begin{bmatrix}1&1\\2&2\\7&10\\3&3\end{bmatrix}$$

They are (1,1)、(2,2)、(3,3) and (7,10).

Theorem 3. For sequences *X* and *Y*, if the identical pair (*i*,*j*) at level *l* has displacements (*k*,*k*), *k*=1,2,…,*w*, where *w* is an integer in [1,4], then the successors of identical pair (*i*,*j*) can be replaced by the successors generated by identical pair (*i*+*w*, *j*+*w*) at level *l*+*w*.

Proof: Since (*i*,*j*) has displacements (*k*,*k*), *k*=1,2,…,*w*, we have $x_{i+k}=y_{j+k}$ for *k*=1,2,…*w*, namely the subsequence "$x_ix_{i+1}x_{i+2}\ldots x_{i+w}$" in *X* is identical to the subsequence "$y_jy_{j+1}y_{j+2}\ldots y_{j+w}$" in *Y*. By Theorem 3, we know the successors of identical pair (*i*,*j*) can be replaced by the successors generated by identical pair (*i*+*w*, *j*+*w*) at level *l*+*w*.

Q.E.D.

Example 5 In Example 2, the displacements of the identical pair (1,2) at level 1 are (1,1)、(2,2)、(3,3) and (7,10). Since it satisfies the condition of Theorem 6 and *w*=3, the successors of (1,2) can be replaced by the successors generated by identical pair (1+3,2+3)=(4,5) at level 4.

Again, the displacements of (4,5) at level 4 are

$$(4,5)\rightarrow\begin{bmatrix}2&2\\3&3\\4&7\\1&1\end{bmatrix}$$

Since it also satisfies the condition of Theorem 6 and *w*=3, the successors of (4,5) can be replaced by the successors generated by identical pair (4+3,5+3)=(7,8) at level 7. The successors of (7,8) are

$$(7,8)\rightarrow\begin{bmatrix}9&11\\10&9\\8&12\\11&10\end{bmatrix}\rightarrow\begin{bmatrix}(9,11)\\(10,9)\\(8,12)\\(11,10)\end{bmatrix}$$

Therefore, the we can skip from (1,2) at level 1 directly to its successors (9,11)、(10,9)、(8,12) and (11,10) at level 8.

Theorem 4. Suppose (i,j) is an identical pair of sequences X,Y at level l. If (i,j) has displacements (1,1) and (r_k, s_k), k=1,2,3, and $w=\min_{1\le k\le 3}(\min(r_k,s_k))$, then the successors of identical pair (i,j) can be replaced by the successors generated by identical pair $(i+w-1, j+w-1)$ at level $l+w-1$.

Proof: Let the character corresponding to the displacement (1,1) be CH, namely, $x_{i+1}=y_{j+1}$=CH. Since $w=\min_{1\le k\le 3}(\min(r_k,s_k))$, the distance between (i,j) and the first character other than CH after (i,j) is w. Therefore, the characters in subsequences "$x_ix_{i+1}x_{i+2}\ldots x_{i+w-1}$" and "$y_jy_{j+1}y_{j+2}\ldots y_{j+w-1}$" are all equal to CH. By Theorem 3, the successors of identical pair (i,j) can be replaced by the successors generated by identical pair $(i+w-1, j+w-1)$ at level $l+w-1$.

Example 6 Let X="TAAAAACGA",

Y= "AAAAAGAC",

TX is：

i	CH(*i*)	0	1	2	3	4	5	6	7	8	9
1	A	2	2	3	4	5	6	9	9	9	–
2	C	7	7	7	7	7	7	7	–	–	–
3	G	8	8	8	8	8	8	8	8	–	–
4	T	1	–	–	–	–	–	–	–	–	–

SX is：

i	CH(*i*)	0	1	2	3	4	5	6	7	8	9
1	A	2	1	1	1	1	1	3	2	1	∞
2	C	7	6	5	4	3	2	1	∞	∞	∞
3	G	8	7	6	5	4	3	2	1	∞	∞
4	T	1	∞	∞	∞	∞	∞	∞	∞	∞	∞

TY is：

i	CH(*i*)	0	1	2	3	4	5	6	7	8
1	A	1	2	3	4	5	7	7	–	–
2	C	8	8	8	8	8	8	8	8	–
3	G	6	6	6	6	6	6	–	–	–
4	T	–	–	–	–	–	–	–	–	–

SY is：

i	CH(*i*)	0	1	2	3	4	5	6	7	8
1	A	1	1	1	1	1	∞	∞	∞	∞
2	C	8	7	6	5	4	3	2	1	∞
3	G	6	5	4	3	2	1	∞	∞	∞
4	T	∞	∞	∞	∞	∞	∞	∞	∞	∞

The displacements of the identical pair (2,1) at level 1 are:

$$(2,1)\rightarrow\begin{bmatrix}1&1\\5&7\\6&5\\\infty&\infty\end{bmatrix}$$

(1,1) is one of its displacement, and the minimum of the elements in the other displacements is w=5. By Theorem 5, the successors of (2,1) can be replaced by the successors generated by identical pair (2+4, 1+4) =(6,5) at level 5. The successors of (6,5) are

$$(6,5)\rightarrow\begin{bmatrix}9&7\\7&8\\8&6\\-&-\end{bmatrix}\rightarrow\begin{bmatrix}(9,7)\\(7,8)\\(8,6)\end{bmatrix}$$

Therefore, the we can skip from (2,1) at level 1 directly to its successors (9,7)、(7,8) and (8,6) at level 6.

5. Framework Of The Algorithm And Complexity Analysis

Based on the operations of generating the successors of the identical character pairs using successor tables, using the pruning and skipping technologies, we present a fast parallel longest common subsequence algorithm named FAST_LCS. The algorithm consists of two phases: the phase of searching for all the identical character pairs and the phase of tracing back to get the longest common subsequences. The first phase begins with the initial identical character pairs, then continuously searches for successors using the successor tables. In this phase, the pruning and skipping technologies are implemented so as to reduce the search space and speed up the process of searching. In the algorithm, a table called *pairs* is used to store the identical character pairs obtained in the algorithm. In the table *pairs*, each record takes the form of (*k*, *i*, *j*, *level*, *pred*, *disp*, *state*) where the data items denote the index of the record, the identical character pair (i,j), its level, the index of its direct predecessor, the distance of skipping from its predecessor and its current state. Each record in *pairs* has two states. For the identical pairs whose successors have not been searched, it is in *active* state, otherwise it is in *inactive* state. In every step of search process, the algorithm searches for the successors of all the identical pairs in *active* state in parallel. Repeat this search process until there is no identical pair in *active* state in the table. In the second phase, the tracing back starts from the identical pairs with the maximal level in the table, and traces back according to the *pred* of each identical pair. This tracing back process ends when it reaches an initial identical pair, and the trail indicates the longest common subsequence. If there are more than one identical pairs with the maximal level in the table, the tracing back procedure for those identical pairs can be carried out in parallel and several longest common subsequences can be obtained concurrently. The framework of algorithm FAST_LCS is as follows:

Algorithm-FAST_LCS (*X,Y*)

Input *X* and *Y*: Sequences with lengths of *m* and *n* respectively;

Output LCS : The longest common subsequence of *X*,*Y*;

Begin

1. Build tables *TX, TY, SX* and *SY*;
2. Find all the initial identical character pairs: (*TX*(*k*, 0),*TY*(*k*, 0)), *k*=1,2,3,4;
3. Add the records of the initial identical pairs (*k*, *TX*(*k*, 0),*TY*(*k*, 0), 1, ϕ, 0, *active*), *k*=1,2,3,4 to the table *pairs*.

   /* For all the initial identical pairs, their *level*=1, *pred*= ϕ ,*disp*=0, and *state=active**/

4. Repeat

   4.1 For all *active* identical pairs *IP*=(*k*, *i*, *j*, *level*, *pred*, *disp*, *active*) in *pairs* parallel-do

   4.1.1 If *IP* satisfies the conditions of skipping then

   4.1.2 *w*= the displacement it can skip

   4.1.3 else *w*=1;

   4.1.4 Produce all the successors of *IP*.

   4.1.5 For each identical pair (*g*, *h*) which is a successor of *IP,* a new record (*k'*, *g*, *h*, *level*+*w*, *k*, *w*, *active*) is generated and inserted into the table *pairs*.

   4.1.6 Change the state of *IP* into *inactive*.

   4.2 End for

   4.3 Use prune operation on all the identical pairs newly produced to remove all the redundant identical pairs from table *pairs*.

5. Until there is no record in *active* state in table *pairs*.
6. Compute *r*= the maximal level in the table *pairs*.
7. For all the identical pairs (*k*, *i*, *j*, *r*, *l*, *disp*, *inactive*) in *pairs* parallel-do

   7.1 *pred* = *l*; $LCS(r) = x_i$.

   7.2 While *pred* $\neq \phi$ do

   7.1. 1 get the p*red*-th record (*pred*, *g*, *h*, *r*', *l*', *disp*', *inactive*) from table *pairs*;

   7.1.2 *pred* = *l*';

   7.1.3 for *j*=1 to *disp*' do $LCS(r'-j-1)=x_{g-j}$

   7.3 end while

8. End for

End.

Assume that the number of the identical character pairs of *X*, *Y* is *L*. In our algorithm, because of the pruning and skipping technologies, the operation of producing successors can be implemented at most once on each identical character pair. Therefore the time complexity for sequentially executing of the algorithm FAST_LCS (*X*, *Y*) is *O*(*L*). Since the table *pairs* has to store all the identical character pairs, it requires *O*(*L*) memory space. Considering that the memory space costs of *TX*, *SX* and *TY*, *SY* are 8*(*n*+1) and 8*(*m*+1), the storage complexity of our algorithm is max{8*(*n*+1)+8*(*m*+1),*L*}. In parallel implementation of the algorithm, since the process for each identical pair can be assigned on one processor, all the process on the identical pairs can be carried out in parallel. Therefore, the time required for the parallel computation is equal to the maximal level of the identical pairs. By Lemma 1, we know that the length of the longest common subsequence of *X*, *Y*, |LCS (*X*,*Y*)|, is equal to the largest level of the identical pairs. Therefore the time complexity of parallel computing is *O*(|LCS(*X*,*Y*)|).

6. EXPERIMENTAL RESULTS

We test our algorithm FAST_LCS on the rice gene sequences of *tigr*[29]database and compare the performance of FAST_LCS with that of Smith－Waterman algorithm[30] and FASTA algorithm[31-32] which are currently the most widely used LCS algorithms. Since both our algorithm and Smith-Waterman's can obtain exactly correct solution, we compare the computation speed of our algorithm FAST_LCS with that of Smith-Waterman algorithm. Also, we compare the precision of our algorithm with that of FASTA using the same computation time.

Table1 compares the computation speed of FAST_LCS with that of Smith-Waterman algorithm on groups of gene sequences pairs with different lengths. Since a test on one pair of sequences takes very short time, it is hard to compare the speed of the algorithms using a single pair of sequence. Therefore we test the algorithms on groups of sequences pairs with similar lengths. We test five groups of sequences pairs each of which consists of 100 pairs of sequences. The total time for each group by the two algorithms are listed in Table1.

Table 1 Comparison of computation speed of fast-lcs with that of smith-waterman algorithm

Name of Sequences	Length *l*	Number of pairs	Time of FAST_LCS (S)		Time of S-W algorithm (S)	
			Total time	Average time	Total time	Average time
gi|21466196～gi|21466195 … gi|21466168～gi|21466167 gi|21466166～gi|30250556 gi|30230255～gi|30230254 gi|30229613～gi|30229612 … gi|30229449～gi|30229448	0⩽*l*⩽ 50	100	0.43	0.0043	1.09	0.0109
gi|30229047～gi|30229046 … gi|30229001～gi|30229000 gi|30228999～gi|30228998 … gi|30228849～gi|30228848	50⩽*l* ⩽100	100	5.57	0.0557	11.55	0.1155
gi|30229447～gi|30229446 … gi|30229249～gi|30229248	10 ⩽ *l* ⩽150	100	29.21	0.2921	65.95	0.6595
gi|30228846～gi|30228845 … gi|30228648～gi|30228647	15 ⩽ *l* ⩽200	100	93.78	0.9378	172.21 3	1.7213
gi|30229247～gi|30229246 … gi|30229049～gi|30229048	20 ⩽ *l* ⩽250	100	230.51	2.3051	425.16	4.2516

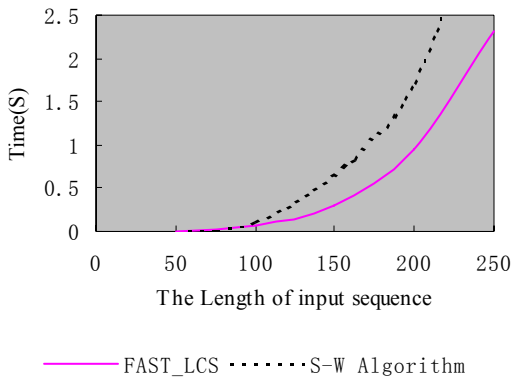

Fig.1 Comparison of the computation time of FAST_LCS with that of Smith－Waterman algorithm

Fig.1 shows the comparison of the computation time of our algorithm with that of Smith－Waterman algorithm. From the table and the figure, we see that our algorithm is obviously faster than Smith－Waterman algorithm for sequences sets of all different lengths. The difference of the computation time between the two algorithms grows faster when the length of sequences become greater than 150. This means our algorithm is much faster and more efficient than Smith-Waterman's for LCS problem of long sequences.

We also compare the precision of our algorithm with that of FASTA on the premise of the same computing time. Here precision is defined as:

$$\text{Precision} = \frac{\text{Length of the common subsequence computed by the algorithm}}{\text{Length of the longest common subsequence in correct match}}$$

From Fig.2, we can see that our algorithm can obtain exactly correct result no matter how long the sequence could be, while the precision of FASTA declines when the length of the sequences is increased. Therefore the precision of our algorithm is much higher than that of FASTA algorithm.

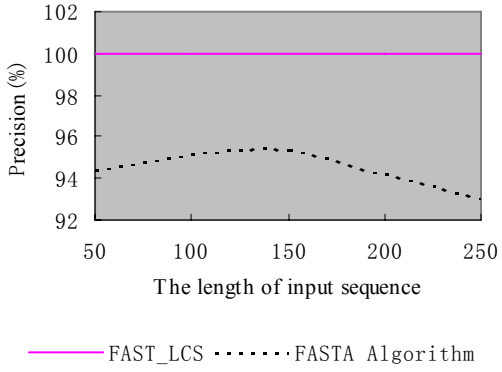

Fig.2 Comparison of the precision of FAST_LCS with that of FASTA using the same computation time

7. CONCLUSION

On the premise of guaranteeing precision of the results of LCS, we present a parallel longest common subsequence algorithm named FAST_LCS based on the identical character pair to improve the speed of LCS problem. Our algorithm first seeks the successors of the initial identical character pairs according to a successor table to obtain all the identical pairs and their levels. Then by tracing back from the identical character pair at the largest level, the result of LCS can be obtained. For two sequences X and Y with lengths n and m, the memory required for FAST_LCS is $\max\{8*(n+1)+8*(m+1), L\}$, here L is the number of identical character pairs and time complexity of parallel implementation is $O(|\text{LCS}(X,Y)|)$, here, $|\text{LCS}(X,Y)|$ is the length of the LCS of X,Y. Experimental result on the gene sequences of *tigr* database shows that our algorithm can get exactly correct result and is faster and more efficient than other LCS algorithms.

8. ACKNOWLEDGMENTS

This research was supported in part by the Chinese National Natural Science Foundation under grant No. 60673060, Chinese National Foundation for Science and Technology Development under contract 2003BA614A-14, and Natural Science Foundation of Jiangsu Province under contract BK2005047.

9. REFERENCES

[1] Bailin Hao，Shuyu Zhang.The manual of Bioinformatics .Shanghai science and technology publishing company，2000：171－172P.

[2] Translated by Yanda Li, Zhirong Sun et.al. Bioinformatics—The practice guide for the analysis of gene and protein, Tsinghua university publishing company,2000：138.

[3] Edmiston E W, Core N G, Saltz J H, et al. Parallel processing of biological sequence comparison algorithms. *International Journal of Parallel Programming*, 1988, 17(3): 259-275.

[4] Lander E. Protein sequence comparison on a data parallel computer. *Proceedings of the 1988 International Conference on Parallel Processing,* 1988. 257-263.

[5] Galper A R, Brutlag D L. Parallel similarity search and alignment with the dynamic programming method. *Technical Report, Stanford University*, 1990.

[6] Smith T.F.,Waterman M.S. Identification of common molecular subsequence. *Journal of Molecular Biology*,1990,215:403-410.

[7] Needleman, S.B. and Wunsch, C.D., A general method applicable to the search for similarities in the amino acid sequence of two proteins, *J. Mol. Biol.*, 48(3):443-453, 1970.

[8] A. Aho, D. Hirschberg, and J. Ullman, Bounds on the Complexity of the Longest Common Subsequence Problem, *J. Assoc. Comput. Mach.*, vol. 23, no. 1, pp. 1-12, Jan. 1976.

[9] O. Gotoh, An improved algorithm for matching biological sequences, *J. Molec. Biol.* 162 (1982) 705-708.

[10] E. W. Mayers, W. Miller, Optimal Alignment in Linear Space, *Comput. Appl. Biosci*. 4(1) (1998) 11-17.

[11] D. S. Hirschberg, A Linear Space Algorithm for Computing Maximal Common Subsequences, *Commun. ACM* 18 (6) (1975) 341-343

[12] Y. Pan, K. Li, Linear Array with a Reconfigurable Pipelined Bus System – Concepts and Applications, *Journal of Information Science* 106 (1998) 237-258.

[13] Jean Frédéric Myoupo, David Seme, "Time-Efficient Parallel Algorithms for the Longest Common Subsequence and Related Problems." Journal of Parallel and Distributed Computing 57(2): 212-223 (1999).

[14] L. Bergroth, H. Hakonen, and T. Raita, "A survey of longest common subsequence algorithms," *Seventh International Symposium on String Processing Information Retrieval*, pp. 39–48, 2000.

[15] A. Aggarwal and J. Park, Notes on Searching in Multidimensional Monotone Arrays, Proc. 29th Ann. *IEEE Symp. Foundations of Comput. Sci.* 1988, pp. 497-512.

[16] A. Apostolico, M. Atallah, L. Larmore , and S. Mcfaddin, Efficient Parallel Algorithms for String Editing and Related Problems, *SIAM J. Computing,* vol. 19, pp. 968-988, Oct. 1990.

[17] M. Lu, H. Lin, Parallel Algorithms for the Longest Common Subsequence Problem, *IEEE Transaction on Parallel and Distributed System*, vol 5. No. 8, August 1994.

[18] K. Nandan Babu, Wipro Systems, and Sanjeev Saxena, Parallel Algorithms for the Longest Common Subsequence Problem , *4th International Conference on High Performance Computing*, December 18-21, 1997 - Bangalore, India.

[19] Y. Robert, M. Tchuente, A Systolic Array for the Longest Common Subsequence Problem, *Inform. Process. Lett.*21 (1985) 191 – 198.

[20] J. H. Chang, O.H. Ibarra, M.A. Pallis, Parallel Parsing on a one-way array of finite-state machines, *IEEE Trans. Computers* C-36 (1987) 64-75.

[21] Guillaume Luce, Jean Frédéric Myoupo: Systolic-based parallel architecture for the longest common subsequences problem. *Integration,* 25(1): 53-70 (1998).

[22] V. Freschi and A. Bogliolo, "Longest common subsequence between run-length-encoded strings: a new algorithm with improved parallelism, "Information Processing Letters, Volume 90 , Issue 4 (May 2004), pp. 167-173 .

[23] Lipman DJ, Altschul SF, Kececioglu JD: A tool for multiple sequence alignment. *Proc.Natl. Acad. Sci.* USA 86, 4412-4415 (1989).

[24] Carrillo H, Lipman DJ: The multiple sequence alignment problem in biology. *SIAM J. Appl. Math.* 48, 1073-1082 (1988).

[25] Stoye J, Moulton V, Dress AW: DCA: an efficient implementation of the divide-andconquer approach to simultaneous multiple sequence alignment. *Comput. Appl. Biosci.*13(6), 625-6 (1997).

[26] Reinert K, Stoye J, Will T: An iterative method for faster sum-of-pair multiple sequence alignment. *Bioinformatics* 16(9),808-814 (2000).

[27] Thompson, JD, Higgins, DG and Gibson, TJ (1994) CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position specific gap penalties and weight matrix choice. *Nucleic Acids Research,*1994, vol.22,No.22.4673-4680.

[28] Feng D-F, Doolittle RF: Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *J. Mol. Evol.* 25,351-360 (1987)

[29] http://www.tigr.org/tdb/benchmark

[30] Altschul, S.F., Gish, W., Miller, W., Myers, E.W., and Lipman, D.J., Basic local alignment search tool, *J. Mol. Biol.*, 215:403-410, 1990.

[31] http://alpha10.bioch.virginia.edu/fasta_www/cgi/

[32] http://www.ebi.ac.uk/services/