Research on Longest Common Subsequence Fast Algorithm

Jiamei Liu,Suping Wu
School of Mathematics and Computer Science
Ningxia University
Yinchuan Ningxia,China

Abstract—*In order to improve the efficiency of searching the longest common subsequence(LCS)， a method of finding LCS(here,the length of the LCS p is much smaller than the length of smaller string of two strings m) is realized in this paper, which transform this problem into solving the problem of matrix L (p, m), by theorem the process of computing each element in the matrix is optimized. Algorithm analysis and experimental results show this algorithm is better than dynamic programming method. And a parallel algorithm based on OpenMP is provided in this paper, which speed of sloving large-scale data is greatly improved.*

Keywords— longest common subsequences; parallel ; OpenMP

I. INTRODUCTION

The longest Common Subsequence (LCS) is the same longest string contained in the two given strings, which delete some elements respectively. The problem of LCS trys to find the two character strings' longest common subsequence. The algorithm of LCS is widely used, such as in the field of gene engineering, it may compare DNA of patients with that of healthy ones, then find out the same parts and different parts so as to do further analysis. The tide of information digitization made plagiarism more and more convenient. This phenomenon is evidently displayed in students' work, so it is urgent to find a highly efficient Plagiarism detection method to help teacher carry out supervision on students. By comparing the similarity between different texts, the detection technique may be realized by LCS algorithm. Reference [1] gives us the application of LCS in the test of text entry. Reference [2] gives us the application of LCS algorithm in program code similarity measurement. Reference [3], Feng Lin and his partners applied LCS algorithm in computer simulation field, which tried to search the human movement sequence. Reference [4] illustrates an algorithm description based on the classic dynamic programming algorithm with time complexity of O(mn). This paper uses one way to solve the problem of LCS, which transforms the original problem of seeking the longest common character string into solving matrix L(p,m)[5], which was realized in a parallel method with the multithreading application program besed on OpenMP. The experiment was taken under two kinds of different hardware environment: the first group has Intel(R) Core(TM)^2 Quad processors, 1G memory and the other group with Intel(R) Core(TM)^2 Quad processors, 4G memory. Both of them have the same windows XP operating system.

II. SERIAL ALGORITHM

A. Definitions and Theorems[5]

Definition 1 Subsequence : Given a string A = A [1] A [2] ... A [m], (A [i] is the i-th letter, A[i]∈ the character set Σ, l<= i<m = $|A|$， $|A|$ is the length of string A), string B is the sub-sequence of A, which refers to B=A[i_1]A[i_2]…A[i_k], where $<i_2<\ldots<i_k$ and k<=m.

Definition 2 Common Subsequence : Given character string A, B, C, C is called the common subsequence of A and B, which refers to C is not only the subsequence of A, but also the subsequence of B.

Definition 3 Longest Common Subsequence (LCS) ： Given string A, B, C, C is called the LCS of A and B, which refers to C is common subsequence of A and B, for any common subsequence D and $|D|<=|C|$

Given string A and B, $|A|$=m， $|B|$=n, we may assume m <= n, the problem of LCS is solving LCS of A and B.

Definition 4 Given a string A=A[1]A[2]…A[m] and the other one, B=B[1]B[2]…B[n], A (1: i) means continuous sequence A A[1]A[2]…A[i], in the same way B (1: j) means continuous sequence B, B[1]B[2]…B [j]. Li (k) refers to that the string A (1: i) and B (l: j) has the LCS with length k, which is the minimum value of j. With the formula is Li (k) = Minj (LCS (A (1: i), B (l: j)) = k).

Theorem 1 ∀ i∈ [1， m]， there is Li(l)<Li(2)<Li(3)<… <Li(m).

Supported by National Natural Science Foundation of China(60963004)
Contact author: Suping Wu

978-1-61284-459-6/11/$26.00 ©2011 IEEE

Theorem 2 ∀ i∈ [1， m-1]， ∀ k∈ [1， m]， there is $L_{i+1}(k) <= L_i(k)$.

Theorem 3 ∀ i∈ [1， m-1]， ∀ k∈ [1， m-l]， there is $L_i(k) < L_{i+1}(k+l)$.

The above three theorems take no account of the situation that Li(k) is not defined.

Theorem 4 If $L_{i+1}(k)$ exists， then its value must be： $L_{i+1}(k)=\text{Min}(j, L_i(k))$. The j is the least integer which meet the following conditions: A[i+l]=B[j] and j> L_i (k-1)。

The proof of theorem l, 2,3,4 can be found in [6] and [7].

B. Algorithmic Idea

Based on the definition and theorem above, we get the recursive formula to compute Li (k). Recurrence relation expressed in matrix form is shown in Figure 1.1. In the matrix, L (ki) = Li (k), where 1 <i <= m, 1 <k <= m; null means L (k, i) does not exist; When i <k, clearly L (k, i) does not exist. Let p = Max (k) (L (k, m) null), it is not hard to prove the diagonal in matrix, L (p, m) , L (1, m-p +1), L (2, m -p +2) ... L (p-1, m-1), L (p, m) corresponding to the sub-sequence B [L (1, m-p +1)] B [L (2, m-p +2)] ... B [L (p, m)] is the LCS of A and B with length p. So, LCS problem is transformed into how to get the matrix. When we encounter i <k, L (k, i) = null and L (k, i) = k, L (k, i +1) = L (k, i +2) = ... L (k, m) = k.

	i=1	2	3	…	m
k=1	L(1,1)	L(1,2)	L(1,3)	…	L(1,m)
2	null	L(2,2)	L(2,3)	…	L(2,m)
3	null	null	L(3,3)	…	L(3,m)
…	…	…	…	…	…
p	null	null	null	null	L(p,m)
p+1	null	null	null	null	null

Figure 1. Solving the matrix L(p,m) of LCS

Now, an example is performed to test the correctness of this algorithm. Given two strings A and B, A=acdabbc,B=cddbacaba,(m=$|A|$=7， n=$|B|$=9)。With the recursive formula given by Theorem, we can easily compute the matrix L of A and B. The result is shown in Fig. 2, where $ denotes ‘null’.

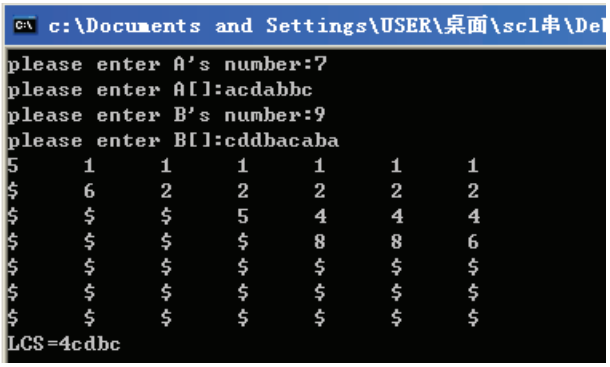

Figure 2

LCS of A and B is B[1]B[2]B[4]B[6], i.e. ‘cdbc’, the length of LCS is 4。

C. Algorithm Pseudo-code

ALGORITHM L(A,B)

INPUT String A and String B

OUTPUT LCS of A and B

Begin

(1) Initialize matrix elements L (k, m) ;

(2) According to the definitions and theorems, obtain the first line elements of L (1, m);

(3) *A*ccording to theorem conclusion

$L_{i+1}(k)=\text{Min}(j, L_i(k))$， compute each remaining element of the matrix line by line. During the solving process, the following two conditions should be utilized to optimize :

(a) if i<k then L(k,i)=null；

(b) if L(k,i)=k then L(k,i+1)=L(k,i+2)=…L(k,m)=k；

(4) If each element in the line p+1 , exit the loop；

(5) After getting matrix L (k, m) , B [L (1, m-p +1)] B [L (2, m-p +2)] ... B [L (p, m)] is the LCS, which length is p, of A and B.

End

D. Algorithm Analysis and Experimental Results

1) Algorithm Analysis

According to theorem, we solve those elements in the first line firstly, i.e. L (1,1) L (1,2), ... L (1, m). Secondly, we solve the second line, and so on, until we discover each element in line p+1 is null. Time complexity of each line is O (n), then the time complexity of entire algorithm is O(pn). So when LCS obtained the number p <<m, to some extent, the speed will be much higher than the dynamic programming. In the process of solving L matrix, instead of storing the entire matrix, we just

store the current line and the previous line. So the space complexity is O (m + n).

2) Experimental Results

Experimental environment: Intel(R) Core(TM) 2 Dual-core processor、1G memory。Software environment:Windows XP

TableⅠ is the time comparison between Dynamic Programming (DP) and algorithm in this paper when the longest common subsequence number is p <<m. Time unit is millisecond.

TABLEI.

Time (ms)	Number of characters		
	m=20	*m=50*	*m=100*
DP algorithm	16[a]	47	63
Algorithm in this paple	0	0	0

From the experimental results, we can see p <<m, this algorithm is superior to dynamic programming obviously.

III. PARALLEL ALGORITHM

A. OpenMP

OpenMP standard was formed in 1997, which is an API, used for writing transplantable multithreaded application. Originally, it's a standard based on Fortran. And later, OpenMP also offered support to C and C++. OpenMP programming model provides a set of instructions, guidance commands, function calls and environment variables, which are independent of platform. It can guide the compiler how and when to use parallelism in the application. For many cycles, it can be inserted with an instruction before the start, to make it execute in parallel. Developers do not need to care about substantial implementation details, which is the work of compiler and OpenMP. Developers only need to consider which cycle should be executed in the manner of multithreading, and how to reconstruct algorithm in order to get better performance on multi-core processor[8].

B. Parallel Idea

Know by theorem 4, $L_{i+1}(k)=\text{Min}(j, L_i(k))$, the most time-consuming operation to get the public string is to compute the matrix L, which are mainly concentrated in the for loop calculations and can use the OpenMP compiler guidance statements to realize the parallelization expediently. Because the process of calculating each column in the loop for solving matrix L (p, m) is not relevant, and the given value of m is a constant in every calculation, we can use "pragma omp parallel for" statements to realize the parallelization directly. In order to determine the number of threads, we must consider the following two points: (1) when the cycle index is small, if we run the cycle with many threads, the entire running time may be longer than the condition with less ones, and increase the energy consumption. (2) If the number of threads you set is much larger than the number of CPU core, there will be a lot of task switching and overhead schedule, and also will reduce the overall efficiency[9].

C. Parallel Algorithm

In this paper, we use OpenMP and C++ language to realize parallel algorithm. And the following give us the parallel algorithm pseudo-code of the fast solution to the longest common subsequence problem:

ALGORITHM L(A,B)

INPUT String A and String B

OUTPUT LCS of A and B

Begin

(1) Parallel initialization matrix elements L (k, m) ;

(2) According to the theorem, obtain the first line elements of L (1, m)

(3) *A*ccording to theorem conclusion $L_{i+1}(k)=\text{Min}(j, L_i(k))$，compute each remaining element of the matrix line by line. During the solving process, the following two conditions should be utilized to optimize :

```
if(i<k)
{
L[k][i]=N; //i<k 时,L(k,i)=null
}
if(L[k][i]==k)
{
 for(l=i+1;l<=m;l++)
       L[k][l]=k;
      }
#pragma omp parallel for
      for(j=1;j<=n;j++)
      {
      if(A[i+1]==B[j]&&j>L[k-1][i])
       {
              L[k][i+1]=(j<L[k][i]?j:L[k][i]);
                     break;
       }
}
```

(4) If each element in the line p+1 , exit the loop；

(5) After getting matrix L (k, m) , B [L (1, m-p +1)] B [L (2, m-p +2)] ... B [L (p, m)] is the LCS, which length is p, of A and B.

End

D. Conclusion Analysis

1) Experimental Results

Experimental environment: (1) Intel(R) Core(TM)2 Dual-core processor、1G memory; (2) Intel(R) Core(TM)2 Quad-cores processors 、 4G memory. Software environment: Windows XP。TableII gives three experimental results of serial algorithm and parallel algorithm, the time unit is ms. Figure 3 shows running time comparison chart.Table II show the time consumption comparison of serial algorithm and parallel algorithms.

TABLEII.

Time (ms)	Number of characters		
	m=300	*m=3000*	*m=6000*
Quad-core processor	0	7203	57225
Dual-core processor	16	10047	78547
Serial	16	10703	92625

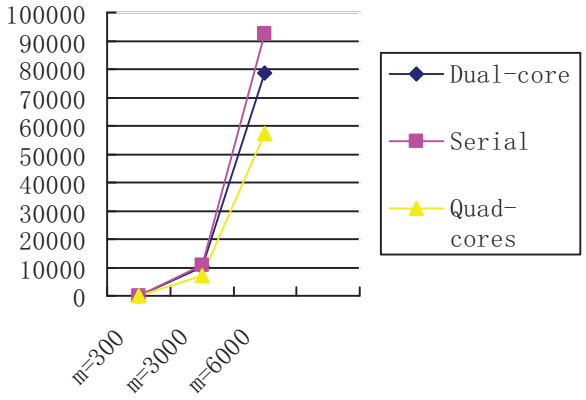

Figure3

2) Results Analysis

When m>300, the time consumption of parallel algorithm is shorter than serial algorithm, as m is growing bigger and bigger, the time difference is bigger. But when m<300, the time of parallel algorithm is longer, because the time consumption of parallel expense acounting for a large proportion, when m increases, the proportion decreases, so the speed of parallel algorithm gets higher and higher. The number of thread can be set, we often acquiesce it is the number of core. As the number of thread increase, the expense will grow, so when the calculation amount is small, the parallel algorithm is not likely a good choise. That is to say parallel algorithm suit, when the calculation amount is relatively big enough. If the number of thread, we set, is far more than the number of core, there will be too much task switching and scheduling expense, which will surely lead to lag the overall efficiency.

IV. CONCLUSION

According L_{i+1} (k)=Min(j, L_i (k)), this paper realized a meathod to find LCS by solving matrix. The algorithm fit the condition when p<<m. This algorithm can improve the rate of sequence match, which is on the premise that it will not affect precision. The algorithm can get a text line as an unit of comparison to accelerate computation. Besides, to accelerate the process much more, this paper offers a new parallel algorithm solving the most time-consuming matrix L(p,m) with OpenMP. In the future, we may attempt to put MPI and OpenMP together in parallel programming, there are still much more work need to be done and improved.

ACKNOWLEDGEMENTS

This paper is supported by the national natural science foundation of China (60963004). we would like to express my warmest gratitude to the national natural science foundation for the financial support.We would like to thank my fellow apprentices for your encouragement and selfless help.

REFERENCES

[1] Jia Zhixian, Application of LCS Arithmetic in Typing Text Test. JI SUAN JI YU XIAN DAI HUA, Vol 137, pp.112-114, 2007,1.(In Chinese).

[2] Yu Haiying,Zhao Junlan, Application of Longest Common Subsequence Algorithm in Similarity Measurement of Program Source Codes. Journal of Inner Mongolia University, vol. 39, pp. 225–229, Mar 2008. (In Chinese).

[3] Feng Lin, Li Pu, Sun Tao, Zheng Hu,Retrieval of Human Motion Capture Data Based on Longest Common Subsequence Model. Journal of System Simulation, vol. 27, pp. 225–229, April 2009. (In Chinese).

[4] Wang Xiaodong, Algorithm Design and Analysis [M]. Beijing: Tsinghua University Press, 2008. (In Chinese)

[5] Li Xin,Shu FengdiFast algorithm of longest common subsequence problem,APPLICATION RESEARCH OF COMPUTERS,Vol 2,pp.28-30,2000.(In Chinese)

[6] Nakatsu N., Kambayashi Y., Yajima S: A Longest Common Subsequence Algorithm Suitable for Similar Text Strings. Acta Informatica 18, 171~179(1982).

[7] HirschbergD.S.: Algorithms for the Longest Common Subsequence Problem J.ACM ,Vol.24, No.4, October1977, PP.664-675.

[8] Li Baofeng,Fu Hongyi,Li Tao,Multi-core programming techniques [M].Electronic Industry Press, 2007. (In Chinese)

[9] Chen Wenguang , Wu Yongwei. MPI and OpenMP parallel programming (C language) Michael J. Quinn. Beijing: Tsinghua University Press, 2004. (In Chinese)