

文章编号:1001-9081(2006)06-1422-03

最长公共子序列的快速算法及其并行实现

刘 维¹, 陈 峻^{1,2}

(1. 扬州大学 信息工程学院, 江苏 扬州 225009;

2. 南京大学 计算机软件新技术国家重点实验室, 江苏 南京 210093)

(lchen@yzcn.net)

摘 要: 求生物序列的最长公共子串是生物信息学中最重要的问题之一, 提出了该问题的一个快速算法, 可对所有初始同字符对并行地寻找其后继同字符对, 并记录下相应层次值。最后通过最大层次值回溯得到比对结果。此外, 该算法采用了剪枝技术, 对于明显不能得出最优比对的同字符将中止其后继的搜索。实验结果证明, 本文算法比其他算法速度快、精确度高。

关键词: 生物信息学; 最长公共子串; 同字符对

中图分类号: TP37 **文献标识码:** A

Parallel longest common subsequence algorithm based on pruning technology

LIU Wei¹, CHEN Ling^{1,2}

(1. Department of Computer Science, Yangzhou University, Yangzhou 225009, China;

2. National Key Laboratory of Novel Software Technology, Nanjing University, Nanjing 210093, China)

Abstract: Searching for the Longest Common Substring (LCS) of biosequences is one of the most important problems in Bioinformatics. Smith-Waterman algorithm and FASTA algorithm are currently the most widely used algorithms. FASTA algorithm runs faster than Smith-Waterman algorithm, but Smith-Waterman algorithm can obtain higher precision. A fast algorithm for LCS problem was presented. The algorithm seeks the successors of the identical character pairs in parallel according to a successor table and record their levels. Finally it traces back from identical character pair with the largest level and get the result of LCS.

Key words: bioinformatics; the longest common subsequence; the identical character pair

0 引言

随着基因组计划的实施, 新的分子生物信息数据大量涌现^[1]。因此在生物信息学研究中, 最首要的任务是如何从海量的数据提取出有价值的知识。而“比较”是我们最常见的方法, 通过将研究对象相互比较来寻找对象可能具备的特性。DNA 片断比对^[2]是最常用最经典的手段, 即将未知序列同已知序列进行比较分析为序列或字符串的最长公共子序列 (LCS) 问题。因为人体 DNA 链中的碱基只有 4 种: A, C, G, T。从而该问题实质上就是一个基于字符表 $\Sigma = \{A, C, G, T\}$ 的 LCS 问题。在人类基因组计划的快速进展过程中, DNA 和蛋白质序列数据库的规模正呈指数增加^[3]。伴随着序列数据库的增长, 三维结构数据库也在不断增长, 于是, 在精确度不受影响的前提下, 如何提高序列比对的速度和效率成了一项重要的课题。序列比对的目的是求出给定的序列对之间的距离, 从而为诸如 DNA 分类、聚类、蛋白质的二级结构预测和生物进化树的创建打下基础, 提供了一个相似性度量的基本工具^[4]。人们对于序列比对中的最长公共子序列问题, 已经作了大量的研究工作。文献^[5]等人证明了最长公共子序列问题的串行算法的时间复杂度的下界为 $O(mn)$ ^[6], 使用动态规划设计的算法可以达到时间和空间的复杂度为 $O(mn)$, 文

献^[7]使用文献^[8]提出的技巧在时间复杂度不变的前提下将空间约减到 $O(m+n)$ 。研究表明若对该问题采用并行化的算法^[9-12], 可以加快问题求解速度。本文针对生物序列的比对问题的需要, 提出一种快速的最长公共子序列的快速算法, 该算法通过对两条字符串建立相应的同字符后续表, 随后对于相应的初始同字符对, 并行地在表中逐层地搜索其后继同字符对, 同时在每一层对所得到的后继集合进行剪枝操作。最后由最大层次值的同字符对进行回溯, 依次求得其所有前驱同字符对, 最后得到相应的比对结果。我们对 tigr 数据库 (TDB) 中的基因序列在 MPP 并行处理机深腾 1800 上进行的实验结果证明, 本文算法与其他经典的比对算法相比, 不但能够取得准确的结果, 而且速度有了较大的提高。

1 同字符后续表及其同字符对

设欲比对的字符串分别为 $X = (x_1, x_2, \dots, x_n)$, $Y = (y_1, y_2, \dots, y_m)$, 其中, $x_i, y_i \in \{A, C, G, T\}$ 。在该算法中, 我们首先要对这两条字符串建立对应的同字符后续表。建表过程如下: 定义 $A = CH(1)$, $C = CH(2)$, $G = CH(3)$, $T = CH(4)$, 将 X , Y 的同字符后续表分别记为 TX, TY , 这里, TX 为 $4 * (n+1)$, TY 为 $4 * (m+1)$ 的二维数组。其中: $TX(i, j)$ 为: X 第 j 个位置上下一个为 $CH(i)$ 的字符的位置; $TY(i, j)$ 为: Y 第 j 个位置上

收稿日期: 2005-12-09; 修订日期: 2006-02-13

基金项目: 国家自然科学基金资助项目 (60473012); 国家科技攻关项目 (2003BA614A-14); 江苏省自然科学基金 (BK20005047)

作者简介: 刘维 (1982-), 女, 江苏江阴人, 硕士研究生, 主要研究方向: 算法优化和并行计算; 陈峻 (1951-), 男, 江苏宝应人, 教授, 博士生导师, 主要研究方向: 算法设计和并行计算。

下一个为 $CH(i)$ 的字符的位置;例如:设 $X = "TGCATA"$, $Y = "ATCTGAT"$ 则 TX 及 TY 分别为:

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	-	-	-
2	C	3	3	3	-	-	-	-	-
3	G	5	5	5	5	5	-	-	-
4	T	2	2	4	4	7	7	7	-

其中“-”表示无定义,即: $TX(i, j) = \text{"-"}$ 表示序列在第 j 个字符后无等于 $CH(i)$ 的字符。由此,我们对 $TX(i, j)$ 的定义如下:

定义1 对序列 $X = (x_1, x_2, \dots, x_n)$, 其同字符后续表 TX 定义为:

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

其中 $SX(i, j) = \{k \mid x_k = CH(i), k > j\}$, $i = 1, 2, 3, 4$, $j = 0, 1, \dots, n$ 。

定义2 在 X, Y 中,若有 $X(i) = Y(j)$, 则记 (i, j) 为一同字符对。所有同字符对的集合记为 $S(X, Y)$ 。

定义3 若 $(i, j), (k, l)$ 皆为同字符对,且有 $i < k, j < l$ 则称 (i, j) 为 (k, l) 的一个前驱,或称 (k, l) 为 (i, j) 的一个后继,记为 $(i, j) < (k, l)$ 。

定义4 若设集合 $P(i, j) = \{(r, s) \mid (i, j) < (r, s), (r, s) \in S(x, y)\}$ 为 (i, j) 的所有后继同字符对集合,若有 $(k, l) \in P$, 且不存在 $(k', l') \in P$, 使得: $(k', l') < (k, l)$, 则称 (k, l) 为 (i, j) 的直接后继,记为 $(i, j) < (k, l)$ 。

定义5 设 $(i, j) \in S(x, y)$, 且不存在 $(k, l) \in S(x, y)$, 使得 $(k, l) < (i, j)$, 则称 (i, j) 为初始同字符对,我们定义初始同字符对的层次为 1。

定义6 对任意的同字符对 $(i, j) \in S(x, y)$, 它的层次号 $level(i, j)$ 定义为:

$$level(i, j) = \begin{cases} 1 & \text{if } (i, j) \text{ 为初始同字符对} \\ \max\{level(k, l) + 1 \mid (k, l) < (i, j)\} & \text{otherwise} \end{cases} \quad (2)$$

由以上定义,我们不难看出以下引理:

引理1 记 X, Y 最长公共子序列的长度为 $LCS(X, Y)$, 则 $LCS(X, Y) = \max\{level(i, j) \mid (i, j) \in S(X, Y)\}$ 。

证明 设 X, Y 的一个最长公共子序列所对应的同字符对依次为 $(x_{i_1}, y_{j_1}), (x_{i_2}, y_{j_2}), \dots, (x_{i_r}, y_{j_r})$, 这里 $r = LCS(X, Y)$ 。根据同字符对的定义,这里, (i_1, j_1) 必然为初始同字符对,且必然有关系 $(i_k, j_k) < (i_{k+1}, j_{k+1}), k = 1, 2, \dots, r-1$, 则 (x_{i_k}, y_{j_k}) 的层次 $level$ 即为 k , r 则为最大的层次 $level$, 即 $r = \max\{level(i, j) \mid (i, j) \in S(X, Y)\}$ 。如果不是,则存在 $r' > r$, 且有同字符序列: $(x_{i_1'}, y_{j_1'}), (x_{i_2'}, y_{j_2'}), \dots, (x_{i_{r'}}, y_{j_{r'}})$ 也对

应一种比对,其长度为 $r' > r$, 即相应的 $r' > r$, 这与 $r = LCS(X, Y)$ 的假设相矛盾。

证毕。

2 产生后继及剪枝操作

在我们的算法中,首先对初始同字符对利用同字符后继表产生其所有的直接后继,然后再对这些后继并行地产生其所有直接后继,重复这样的操作,直至不能继续产生后继为止。因此,由同字符对产生其所有直接后继,是本算法的一个基本操作。对某 $(i, j) \in S(x, y)$, 由 (i, j) 产生一组其后继同字符对的操作可表示为:

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

即对 TX 的第 i 列、 TY 的第 j 列相应元素进行配成对偶,例如对上例中的同字符对 $(2, 5)$, 上述操作可表示为:

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

由于 $(3, -)$ 及 $(-, -)$ 不表示一个同字符对,表示这一支搜索不能得出所需的结果,故应舍去。产生的结果说明 $(2, 5)$ 的后继为 $(4, 6), (5, 7)$ 。应该说明的是,所产生的不一定是 (i, j) 的直接后继。例如 $(5, 7)$ 不是 $(2, 5)$ 的直接后继,因为 $(2, 5) < (4, 6) < (5, 7)$, 在这种情况下我们又可以将 $(5, 7)$ 剪去。

又如对于同字符对 $(1, 1)$, 上述操作可表示为:

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

即产生 $(1, 1)$ 的 4 个后继: $(4, 6), (3, 3), (2, 5), (5, 2)$ 后,同上所述,因为 $(3, 3) < (4, 6)$, 我们可以将 $(4, 6)$ 剪去。随后我们在得到 $(3, 3), (2, 5), (5, 2)$ 的三组后继同字符对后,也要进行剪枝操作:

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

即任取某一同字符对 (i, j) , 将该同字符对在本层上与本组及其他组的同字符对作比较,若有 $(k, l) < (i, j)$, 则将 (i, j) 剪去。例如,对于同字符对 $(6, 6)$, 因为它是其他组中的同字符对 $(5, 4)$ 的后继,所以我们将它剪去。又如因为 $(5, 7)$ 是 $(4, 6)$ 的后继,也可将其剪去。这样,我们逐个剪去一些同字符对,直到不能继续剪枝再进行下一轮产生后继的操作。

以上产生后继同字符对及剪枝操作的正确性,可由以下的引理及定理 1 来保证。

引理2 对任一同字符对 (i, j) , 用上述方法可以产生其所有的后继。

证明 根据 (3) 知,由 (i, j) 可以产生 $(TX(k, i), TY(k, j)), k = 1, 2, 3, 4$; 又根据 (1) 可知, $TX(k, i)$ 是 $SX(i, j)$ 中的最小者,即在序列 X 中,位于字符 x_i 之后且距离 x_i 最近的第 k

种同字符 $CH(k)$ 。类似地, $TY(k, j)$ 是指在序列 Y 中, 寻找位于字符 y_j 之后且距离 y_j 最近的第 k 种同字符。又因为 $k = 1, 2, 3, 4$, 即包含了所有可能的字符配对, 所以一次后继产生操作可以得到 (i, j) 的所有直接后继。同理, 对它的每一个直接后继可再进行后继产生操作得到其每一个直接后继的所有的直接后继。依此类推, 可对同字符对 (i, j) 产生其所有的后继。

证毕。

定理 1 如果在同一层上所产生的同字符对 (i, j) 和 (k, l) 中, 有 $(k, l) > (i, j)$, 则剪去 (k, l) 不影响算法得到最优解。

证明 设在上一层产生 (k, l) 的字符对为 (k_1, l_1) , 产生 (i, j) 的字符对为 (i_1, j_1) , 又设经由 (k_1, l_1) 及 (k, l) 所产生的公共子序列为 $a_1 a_2 \cdots a_m a_{m+1} \cdots a_r$, 其中 a_m 对应于 (k_1, l_1) , a_{m+1} 对应于 (k, l) 。设经由 (i_1, j_1) 及 (i, j) 所产生的公共子序列为 $b_1 b_2 \cdots b_m b_{m+1} \cdots b_q$, 其中 b_m 对应于 (i_1, j_1) , b_{m+1} 对应于 (i, j) , 由于 $(k, l) > (i, j)$, 根据引理 2, (k, l) 必然在 (i, j) 后被产生, 则必存在 b_s ($m+1 < s < q$), 使得 b_s 对应于 (k, l) , 由于 $a_m a_{m+1} \cdots a_r$ 及 $b_s b_{s+1} \cdots b_q$ 都是由同字符对 (k, l) 进行若干次后继操作产生的局部最长公共子序列, 因有 “ $a_m a_{m+1} \cdots a_r$ ” = “ $b_s b_{s+1} \cdots b_q$ ”, 即有 $q - s = r - m$, 即: $q = r + (s - m)$ 。因为 $s > m$, 故有 $q > r$ 。这说明了, 经由 (k, l) 在第 m 层上所产生的子序列 “ $a_m a_{m+1} \cdots a_r$ ” 必不是最长公共子序列, 可以在第 m 层将其剪去。

证毕。

3 算法框架及复杂性分析

综上所述, 本文基于同字符对的并行最长公共子序列算法是一个从初始同字符对开始, 利用同字符后续表不断搜索同字符对的后继的过程。在此过程中, 我们采用了剪枝技术, 去掉明显不能得出最优解的搜索分支, 以提高搜索速度。对于搜索到的同字符对, 我们设立一个同字符对表 Table, 表中的每一项由四元组 $(i, j, level, pre)$ 构成, 表示同字符对 (i, j) 及其层次和前驱, 对于表中尚未进行搜索后继操作的同字符对, 将它们置为 active 状态。在每一轮搜索中, 可以并行地对所有处于 active 状态的同字符对同时搜索其后继。搜索过程直至 Table 表中不存在 active 的同字符对为止, 然后对层数最大的同字符对根据其前驱由表 Table 进行回溯可得到最长公共子序列。因此, 算法框架如下:

算法 FAST_LCS(X, Y)

输入: 长度分别为 m, n 的序列 X, Y ;

输出: X, Y 的最长公共子序列 $LCS(X, Y)$;

Begin

构造 TX, TY 表;

找出所有的初始同字符对: $TX(k, 0), TY(k, 0), k = 1, 2, 3, 4$;

对所有初始同字符对赋予 $level = 1$, 前驱为 ϕ , 加入同字符对表 Table 中, 并将它们置 active 状态;

对 Table 中有 active 同字符对 $(i, j, level, pre)$ 并行地进行:

产生 $(i, j, level, pre)$ 的后继集合, (i, j) 退出 active 状态;

后继集合中的 (k, l) 表示为: $(k, l, level + 1, (i, j))$, 随后对本层所产生的所有后继进行剪枝, 剪枝后将 (k, l) 存入 Table 中, 并将 (k, l) 设为 active 状态。

在 Table 中找出层次 level 最大的项, 根据其前驱进行回溯, 继而找出所有同字符对, 构成一种最长公共子序列 $LCS(X, Y)$ 。

End

设 X, Y 中同字符对的个数为 L , 由于本算法要对 X, Y 中所有同字符对进行一次产生后继操作。由于剪枝技术, 对于每一个同字符对不可能重复进行这样的操作, 因此算法 FAST_LCS(X, Y) 串行执行的过程即为对所有同字符对进行一次产生后继操作的过程, 因此算法的时间复杂度为 $O(L)$ 。由于 Table 表记录所需处理的同字符对, 其所占的存储空间为 $O(L)$, 同字符后续表 TX, TY 的存储空间分别为 $4 * (n + 1)$ 及 $4 * (m + 1)$ 。本文算法所需的存储复杂度为 $\max\{4 * (n + 1), 4 * (m + 1), L\}$ 。在并行计算中, 由于对每一层的计算是并行的, 可以在 $O(1)$ 时间内完成, 因其有 $|LCS(X, Y)|$ 层, 并行计算的时间复杂度为 $O(|LCS(X, Y)|)$ 。

4 实验结果及分析

4.1 串行计算结果

我们从文献[13]数据库中抽取了稻谷基因序列进行实验, 并将本文算法分别与 Smith-Waterman 算法^[14~15]及 FASTA 算法^[16~17]在速度上进行了比较, 结果如图 1 所示。

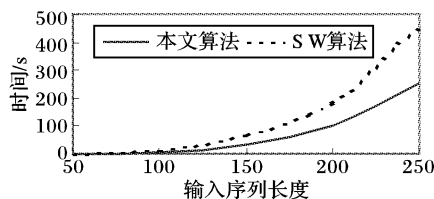


图1 本文算法与 SW 算法时间的比较

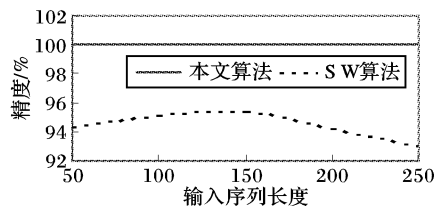


图2 本文算法与 FASTA 算法精度的比较

由图 1 可以看出, 对于不同的序列长度, 本文算法要明显快于 Smith-Waterman 算法。在输入序列长度为 150 之前, 速度变化较为缓慢, 而在序列长度为 150 之后, 速度变化异常迅速, 本文算法的速度超过 SW 算法。本文算法和 SW 算法虽然都能得精确的解, 但本文所需的计算时间要小得多。

我们用本文算法与 FASTA 算法在计算时间相同的情况下, 对计算结果精度 (精度是指算法求得的公共子序列长度与正确匹配中的最长公共子序列长度之比) 进行了比较, 其结果如图 2 所示。由图 2 可见, 不管输入序列长度如何增加, 本文算法都能够取得准确的结果, 精度要明显优于 FASTA 算法。

4.2 并行算法结果

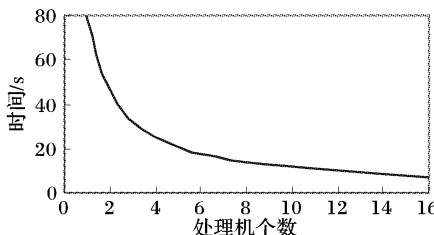


图3 并行计算的时间随处理机个数增加的变化情况

我们将处于 active 的同字符对分配给若干个处理机, 随后每个处理机并行的执行后继产生操作, 我们用本文算法在 MPP 处理机深腾 1800 上采用 MPI 绑定 C 语言编程对该并行算法实现, 实验结果如图 3 所示: 即对于给定长度的生物序

(下转第 1427 页)

表1 实验1中算法采用的具体参数

整体数据集大小 N	数据对象维数 D	样本子集规模 S	抽样次数 J	最大最小距离法参数 m
600	2	60	10	0.35

从图5(a)可以看出,MCA算法可以自动地获取9个聚类中心,在图中分别以黑色实心点表示,以它们为中心的类也分别用虚线圆圈标注,并配有1到9的类标号。其中1、3、6、7、8合并成一个大类,2、4、5、9各自独立成类。MCA算法的最终聚类结果:共聚成了五个类,如图5(b)所示,其中第一个类由1、3、6、7、8五个聚类中心联合代表。采用k-means算法执行20次,聚类结果中14次都出现类似图5(c)陷于局部极小的情况,另6次陷于局部极小的情况各不同。

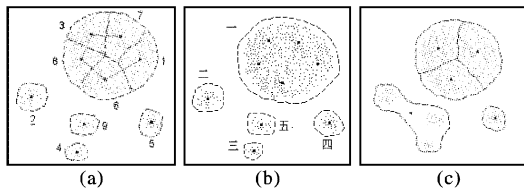


图5 实验1结果对比

当存在簇间差异较大,尤其是有较多小簇的时候,原始k-means算法由于初始聚类中心的选择过于随机,聚类结果相当不稳定,聚类效果不佳。MCA算法则能够更加准确、有效地进行聚类。

3.2 仿真实验2

仿真实验2采用的数据集如图6(a)所示。该数据集由1000个点构成,其显著特点是簇的形状十分不规则,呈延伸状。

运用MCA算法,该数据集自动发现了11个聚类中心,在图6(b)中分别由数字1到11标识出来。合并之后,MCA算法最终产生2个类,每个类各由五个聚类中心联合代表,如图

6(c)所示。采用原始k-means算法进行聚类,执行20次,每次都得出类似于图6(d)陷于局部极小的结果。

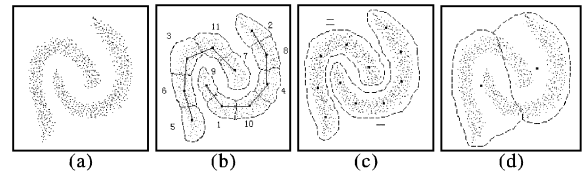


图6 实验2的数据分布及结果对比

可见:原始算法由于采用误差平方和准则函数,算法的每一次迭代都是倾向于发现超球面簇,对延伸等不规则状数据集进行聚类几乎无能为力。MCA算法对于不规则簇有着良好的聚类能力,聚类性能优于原始算法。

参考文献:

- [1] HAN J, KAMBER M. 数据挖掘概念与技术[M]. 范明, 孟小峰, 等译. 北京: 机械工业出版社, 2002. 223 - 262.
- [2] KUMAR M, ORLIN JB, PATEL NR. Clustering data with measurement errors[R]. Technical Report, RRR 12 - 2005, New Jersey: RUTCOR, Rutgers Center for Operations Research, 2005.
- [3] SU MC. A modified version for k-means[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2001, 23(6): 674 - 680.
- [4] FAYYAD U, REINA C, BRADLEY PS. Initialization of iterative refinement clustering algorithms[A]. Proceedings of Fourth International Conference on Knowledge Discovery and Data Mining[C]. Menlo Park: AAAI Press, 1998. 194 - 198.
- [5] CHAUDHURI D, CHAUDHURI BB. A novel multiseed nonhierarchical data clustering technique[J]. IEEE Transactions on Systems, Man and Cybernetics: PartB, 1997, 27(5): 871 - 877.
- [6] 李金宗. 模式识别导论[M]. 北京: 高等教育出版社, 1994.
- [7] 张春阳, 周继恩, 钱权, 等. 抽样在数据挖掘中的应用研究[J]. 计算机科学, 2004, 31(2): 126 - 128.

(上接第1424页)

列,随着处理器个数的增加,计算速度会有大大的提高。虽然图3显示在处理器个数达到6以后,速度变化缓慢,但这是符合并行处理的加速比性能的Amdahl定律的。从总体上讲,随着处理器个数的增加,相应的计算时间会随之减少。

5 结语

为了在不影响精确度的前提下,提高序列比对的速度,本文提出了基于同字符对的求生物序列的最长公共子序列的并行算法。本文使用了同字符对及后继产生操作,并在操作过程中不断更新level值,最终由最大层次值的同字符对进行回溯得到最长公共子序列,由此保证了该算法的精确度。同时本文又使用了同字符后续表及剪枝技术,大大降低了字符搜索的时间,从而提高了速度。

参考文献:

- [1] 郝柏林, 张淑誉. 生物信息学手册[M]. 上海: 上海科学技术出版社, 2000. 171 - 172.
- [2] NEEDLEMAN SB, WUNSCH CD. A general method applicable to the search for similarities in the amino acid sequence of two proteins[J]. J. Mol. Biol., 1970. 443 - 453.
- [3] 生物信息学—基因和蛋白质分析的实用指南[M]. 李衍达, 孙之荣, 等译. 北京: 清华大学出版社, 2000.
- [4] WATERMAN MS. Introduction to Computational Biology, Maps, Sequences and Genomes[M]. London: Chapman & Hall. 1998.
- [5] AHO A, HIRSCHBERG D, ULLMAN J. Bounds on the Complexity of the Longest Common Subsequence Problem[J]. J. Assoc. Com-

put. Mach., 1976, 23(1): 1 - 12.

- [6] GOTOH O. An improved algorithm for matching biological sequences[J]. J. Molec. Biol. 1982, 162: 705 - 708.
- [7] MAYERS EW, MILLER W. Optimal Alignment in Linear Space[J]. Comput. Appl. Biosci. 1998, 4(1): 11 - 17.
- [8] HIRSCHBERG DS. A Linear Space Algorithm for Computing Maximal Common Subsequences[J]. Commun. ACM, 1975, 18(6): 341 - 343.
- [9] 张法, 乔香珍, 刘志勇. 基于Smith-Waterman算法的并行分而治之生物序列比对算法[J]. 中国科学E辑: 技术科学, 2004, 34(2): 190 - 199.
- [10] EDMISTON EW, CORE NG, SALTZ JH, et al. Parallel processing of biological sequence comparison algorithms[J]. International Journal of Parallel Programming, 1988, 17(3): 259 - 275.
- [11] LANDER E. Protein sequence comparison on a data parallel computer[A]. In: Proceedings of the 1988 International Conference on Parallel Processing[C]. 1988. 257 - 263.
- [12] GALPER AR, BRUTLAG DL. Parallel similarity search and alignment with the dynamic programming method[R]. Technical Report, California: Stanford University. 1990.
- [13] <http://www.tigr.org/tdb/benchmark>, 2005.
- [14] SMITH TF, WATERMAN MS. Identification of common molecular subsequence[J]. Journal of Molecular Biology. 1990, 215: 403 - 410.
- [15] ALTSCHUL SF, GISH W, MILLER W, et al. Basic local alignment search tool[J]. J. Mol. Biol., 1990, 215: 403 - 410.
- [16] <http://alpha10.bioch.virginia.edu/fastaweb/cgi/>, 2005.
- [17] <http://www.ebi.ac.uk/services/>, 2005.