基于 Smith-Waterman 算法的并行分而治之生物序列比对算法

张 法 乔香珍

(中国科学院计算技术研究所, 北京 100080)

刘志勇

(国家自然科学基金委员会, 北京 100085)

摘要 生物序列比对是生物信息学中最常见的问题之一, 基于动态规划思想的 Smith-Waterman 算法是序列比对中最基本的算法. 然而现有的并行 Smith-Waterman 算法都需要庞大的内存, 且无法处理大规模的数据串, 随着生物数据的急剧增长, 这些并行算法对内存空间的需求已成为需要迫切解决的问题. 由此提出一种并行生物序列比对算法, PSW-DC 算法, 该算法采用分而治之的方法把 query 序列划分为若干片段, 并分配给相应的各个处理器, 而后并行地按 Smith-Waterman 算法与目标(subject)序列进行比对, 再通过按一定规则的扩展过程求取序列的优化匹配. 与其他并行算法相比, 该算法有效地降低了内存空间的需求, 并实现了对大规模数据串的并行处理. 为实现该算法, 给出了一种称作 C&E 的拓展规则及实现方法. 且该方法已经在实际系统中得到实现.

关键词 生物序列比对 动态规划 分而治之 并行处理 内存空间

生物序列(DNA 序列)可以看作是由一些固定的字符(A, C, G, T)所构成的字符串, 两条序列(序列 S 和 T)的比对可以简单地表示为: 把 S 和 T 这两条序列上下罗列起来, 然后依次比较它们在每一个位置上的相似情况, 从而找出这两条序列间具有最大相似度的子序列片断[1].

关于两条序列的比对问题, 人们从不同的角度提出了很多算法, 其中基于动态规划的算法是目前最基本的一种算法. Needleman 和 Wunsch[2]最早将动态规划

2002-10-08 收稿, 2003-08-01 收修改稿

方法引入到生物序列比对问题中, 提出了 Needleman-Wunsch 算法. Smith 和 Waterman[3]对这一算法进行了改进, 并提出了著名的 Smith-Waterman 算法, 但是该算法的时间复杂度是 $O(mn)$, 其中 m 和 n 分别为两条序列的长度, 另外该算法需要保存一个大小为$(m+1)\times(n+1)$的动态规划矩阵. 后来, Altschul[4,5], Green[1] 等都提出了不同的序列比对算法, 提高了比对的速度, 而且使得算法的空间复杂度有了很大的降低. 然而这些算法速度的提高是以牺牲结果的质量为代价的. 随着越来越多的基因组序列被测定, 一条序列会变得非常庞大, 从而不仅使得算法的运行时间变得非常庞大, 而且对存储容量也提出了极高的要求.

Edmiston[6]等在两种不同体系结构的机器上实现了序列比对的并行处理, 该算法最多可以利用 $O(\min\{m,n\})$个处理器; Lander[7]等在数据并行机 CM2 上对动态规划的算法实现了并行化; Galper[8]等提出了两种不同的 Smith-Waterman 并行策略: Row Wavefront 策略和 Diagonal Wavefront 策略; Qiao[9]概括了动态规划的多种并行策略: 流水线策略、反对角线策略以及块反对角线策略. 然而这些算法需要在一个或多个处理器上存储整个动态规划矩阵, 对处理器存储容量的要求很高, 甚至在某种情况下, 当动态规划矩阵的大小超过了任一个处理器的内存容量时, 这些算法就无法对序列进行处理.

由于考虑到一条序列中每一个碱基或氨基酸与另一条序列中所有碱基或氨基酸的比较, 因此 Smith-Waterman 算法结果的敏感性(质量)与其他算法相比是最高的, 可以用于较远家族序列的相似性比较, 但是由于该算法的运算时间以及对内存空间的需求, 使得无法完成对大规模序列的处理.

生物数据容量的急剧增长对生物信息学提出了严峻的考验. 从序列的拼接、序列的比对、基因区域的预测到蛋白质三维结构和功能的预测, 随着生物数据容量的增长, 传统的算法对内存空间的庞大需求以及漫长的运行时间, 已经无法满足对大规模数据的处理, 因此新算法的研究以及对传统算法的并行化就显得尤为重要. 而对于传统算法的并行化, 如何降低算法对内存的需求以及提高算法的运行速度就成为并行化研究的关键.

针对序列比对算法由于对内存空间的庞大需求而无法对大规模序列进行处理这一问题, 我们采用分而治之的策略, 提出了一种针对 cluster 体系结构的并行生物序列比对算法, PSW-DC 算法. 该算法把 query 序列划分为若干序列片段并分配给相应的各个处理器, 各片段并行地分别与目标序列进行比对, 然后在各片段比对中间结果的基础上采用选择和拓展的方式, 获得最终比对结果. 按照该算法, 每个处理器只需存储原算法所需的内存空间的 $1/p$, 其中 p 为处理器的个数, 从而大大降低了原算法对内存的需求量. 我们又给出了一种从中间结果获得最终结果的选择与拓展规则和归并方法(C&E 方法)来获得最终的比对结果.

1　Smith-Waterman 算法

1.1　生物序列的比对

生物序列比对, 即相似性比较, 是生物信息学中一种最基本的操作, 从序列

1) Green P. http: //bozeman.bvt.washington.edu/phrap/phrap.docs/phrap.html.1996

片段的拼接、基因区域的预测到 DNA 和蛋白质的结构功能预测都需要进行序列相似性的比较. 由于物种在漫长的进化过程中出现的变异情况, 使得原本相同的两条序列出现差异. 因此两条序列比对时, 整条序列之间可能不存在精确的匹配, 但是在某些片段上却可能有着很高的相似性.

在生物信息学中用如下的一些定义来描述序列的比对和相似性[10,11].

记分函数 我们用Σ来表示序列中所有字符的集合(在 DNA 序列中Σ={*A*, *C*, *G*, *T*}), 用'-' 来代表由于变异而造成序列中某一位置缺失的字符, 则记分函数 *f* 可表示为 f: $\Sigma\cup\{\text{'-'}\}\times\Sigma\cup\{\text{'-'}\}\to R$, 即对于任意两个字符 x 和 $y\in\Sigma\cup\{\text{'-'}\}$, $f(x,y)$ 表示字符 x 和 y 进行比较时的分值. 一般情况下, 若 x 和 y 相同时 $f(x, y)$为正值, 否则为零或负值.

空位 在单个序列中任意连续的尽可能长的间隔称为一个空位, 在空位中的间隔('-')个数称为空位的长度, 用*#spaces* 来表示, 序列中空位的个数用*#gaps* 来表示.

例如, 在如下的序列比对中:

$$S' = a\ t\ t\ c\ -\ -\ g\ a\ -\!-t\ g\ g\ a\ c\ c$$
$$T' = a\ -\ -\ c\ g\ t\ g\ a\quad t\quad t\ -\!-\!-\ c\ c$$

#gaps = 4, *#spaces* 分别为 2, 1, 2, 3.

空位的引入是为了补偿由于变异而产生的插入或缺失, 每引入一个空位, 比对的分值都会有所扣除. 通常的空位罚分函数可表示为: $Penalty_{gap} = W_g + W_s \times (\#spaces)$, 其中 $Penalty_{gap}$=为空位的总罚分, W_g 表示为初始化一个空位的罚分, W_s 表示空位延伸一个间隔的罚分.

全局最优比对和局部最优比对 对于两条序列 S 和 T 的全局比对可以用序列 S'和 T'来表示, 其中,

1) S'和 T'分别是在 S 和 T 中加入一些空位而得到的序列;

2) $|S'|=|T'|$, $|S'|$为序列 S'的长度.

将序列 S'和 T'相应的位置进行一一的比较, 其分值 *Score* 可用如下的公式来表示:

$$Score = \sum_{i=1}^{l} f(s_i', t_i') + W_g \times (\#gaps) + W_s \times (\#spaces),$$

其中 $l=|S'|=|T'|$, s_i', t_i' 分别为序列 S'和 T'的第 i 个字符.

序列 S 和 T 的全局最优比对是指在 S 和 T 的所有比对中相似性分值 *Score* 最大时所对应的比对. 序列 S 和 T 的局部最优比对可以用α 和β来表示, 其中α和β分别为 S 和 T 的子序列, 且α和β的 *Score* 分值是 S 和 T 中所有子序列比对分值的最大值. 在生物学的许多应用中局部比对要比全局比对更有实际的意义.

1.2 Smith-Waterman 算法

Smith-Waterman 算法是目前在生物信息学中使用最为广泛的算法之一, 该算法可以大致分为以下两个部分:

1) 利用初始条件和迭代关系式计算两个序列的所有可能的比对分值, 并将结果存放于一个矩阵中.

2) 利用动态规划的方法回溯寻找最优的比对结果.

对于给定的两条序列 $S = s_1, s_2, \cdots, s_n$ 和 $T = t_1, t_2, \cdots, t_m$, 它们对应的长度分别为 n 和 m. 根据动态规划的方法, 我们构造一个大小为 $(n+1)\times(m+1)$的矩阵 D(称作动态规划矩阵), 用来存放所有可能的比对结果. 矩阵 D 可以通过如下的公式计算得到.

①初始条件:
$$D(i,0) = D(0,j) = 0. \tag{1}$$

② 递归关系:
$$D(i,j) = \max\begin{cases} 0, \\ D(i-1,j-1) + f(s_i, t_j), \\ \max\limits_{1\leqslant x\leqslant i-1}\{D(i-x,j) - W_x\}, \\ \max\limits_{1\leqslant y\leqslant j-1}\{D(i,j-y) - W_y\}, \end{cases} \tag{2}$$

其中 $1\leqslant i\leqslant n$, $1\leqslant j\leqslant m$, W_x 为在序列 S 中添加一个长度为 x 的空位的罚分, Wy 为在序列 T 中添加一个长度为 y 的空位的罚分, 其结果可能是以下两种情况中的一种: 在已存在的空位末端添加一些间隔或者重新创建一个空位.

这样 $D(i, j)$就表示为 $s_1, s_2, \cdots, s_i$ 和 $t_1, t_2, \cdots, t_j$ 的最优比对分值. 在动态规划矩阵 D 中找出 i^* 和 j^*, 使得

$$D(i^*, j^*) = \max_{1\leqslant i\leqslant n,\ 1\leqslant j\leqslant m} D(i,j). \tag{3}$$

则 $D(i^*, j^*)$表示序列 S 和 T 的局部最优比对分值. 从 $D(i^*, j^*)$开始, 按照从右下到左上的方向, 对矩阵 D 进行回溯就可以求得局部最优比对序列, 下面这段代码简单描述了回溯的处理过程, 这里仅考虑(2)式中 x 和 y 等于 1 的情况, 因此代码中的 *gapcost* 等于 W_1.

```
i = i*; j = j*; k = 0;
while((i>0) && (j>0) && (D(i, j)>0))
        {if (D(i, j) == D(i−1, j−1) + f(s_i, t_j)
        { s'_k = s_i;  t'_k = t_j; i−−; j−−;}
        else if (D(i, j) == D(i−1, j) −gapcost)
        { s'_k = s_i; t'_k = '-'; i−−;}
        else if (D(i, j) == D(i, j−1)−gapcost)
        { s'_k = '-'; t'_k = t_j; j−−;}
   k++; }
```

这样通过回溯可得到局部最优比对序列 S' 和 T'. Smith-Waterman 算法的时间复杂度为 $O(mn)$, 其中 m 和 n 分别为两条序列的长度. 例如: 序列 $S = a\ g\ g\ c\ t\ a\ g$ 和序列 $T = c\ c\ c\ g\ t\ a$. 定义记分函数为: $f(x, x) = +2$, $f(x, y) = f(x, '-') = f('-', y) = -1$. 通过计算动态规划矩阵 D, 可以得到局部最优比对的分值为 $D(6, 6) = 5$, 从 $D(6, 6)$ 利用回溯算法就可得到局部最优比对序列, 如图 1 所示. 根据图 1 的回溯路径可得到一个局部最优比对: $S' = g\ c\ t\ a$, $T' = g\ '-'\ t\ a$.

由(2)式可知: 在矩阵 D 中元素 $D(i, j)$的计算依赖于元素 $D(i-1, j-1)$, $D(i-x, j)$

和 $D(i, j-y)$的值, 如图 2 所示. 由于数据间的依赖关系, 上文所提到的许多并行算法[1,6~9]都采用并行的方法来计算动态规划矩阵 D. 然而, 随着生物序列数据容量的增加, 这些算法的内存空间需求问题就变得日益严峻. 另外, 针对于 cluster 体系结构的并行系统, 这些并行算法不仅在处理器之间的通信开销较大, 而且每个处理器需要存储整个的动态规划矩阵 D. 因此针对于 cluster 体系结构的并行比对算法的研究就显得尤为重要.

		c	c	c	g	t	a
	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
a 1	0	0	0	0	0	0	2
g 2	0	0	0	0	2	1	1
g 3	0	0	0	0	2	1	0
c 4	0	2	2	2	1	1	0
t 5	0	1	1	1	1	3	2
a 6	0	0	0	0	0	2	5
g 7	0	0	0	0	2	1	4

图 1 Smith-Waterman 算法的矩阵和回溯路径

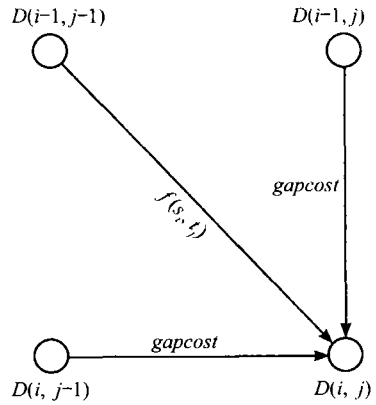

图 2 数据依赖关系

2 基于 Smith-Waterman 算法的并行分而治之比对算法——PSW-DC 算法

考虑到 Smith-Waterman 算法对内存容量的要求, 并结合 cluster 体系结构的特点, 我们提出了一种粗粒度的并行比对算法: PSW-DC 算法. 该算法整体上基于分而治之的策略, 而在各具体片段的比对中采用 Smith-Waterman 算法, 其关键问题在于结合中间结果过程中的选择与拓展规则和方法. 与其他并行 Smith-Waterman 算法相比, 该算法有效地降低了对内存空间的需求.

2.1 PSW-DC 算法

该算法的基本思想是: 采用数据分割的方法给每个处理器分配一定的数据, 然后每个处理器独立地对所分配的数据运行 Smith-Waterman 算法, 并分别得到一个中间比对结果, 最后对各个处理器计算的中间结果进行处理, 得到最终的最优比对结果.

在实际工作中, 我们通常将未知序列称作 query 序列, 将已知序列称作 subject 序列. 给定两条序列: query 序列 S 和 subject 序列 T. 首先根据处理器的个数(用 p 来表示)将 query 序列 S 划分成 p 份子序列, $S_0, S_1, \cdots, S_{p-1}$, 每份子序列的长度为 $|S|/p$ (在划分 query 序列时, 也可以使相邻两份子序列之间存在有一定的重叠, 即每份子序列的长度略微大于 $|S|/p$, 在本文并没有讨论这种情况), 并将子序列 S_i

$(0 \leqslant i < p)$ 分配给处理器 P_i, 同时将 subject 序列 T 广播给所有的处理器; 然后处理器 P_i 对 query 子序列 S_i 和 subject 序列 T 独立运行 Smith-Waterman 算法, 得到 S_i 和 T 的一个最优的局部比对 A_i; 最后, 对每个处理器上的计算结果 A_i 进行处理, 得到 query 序列 S 和 subject 序列 T 的最优局部比对 A. 该算法的示意图如图 3 所示(以 p = 3 为例).

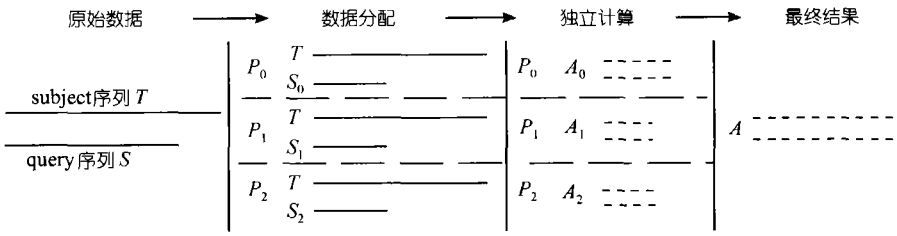

图 3　PSW-DC 算法示意图

从示意图中可以看出每个处理器 P_i 单独运行 Smith-Waterman 算法后, 只需存储一个大小为 $(|S|/p+1)\times(|T|+1)$ 的矩阵 D_i, 对 D_i 进行回溯可求得子序列 S_i 和序列 T 的局部最优比对 A_i, 其分值我们用 M_i 来表示. 那么, 如何根据这些中间比对结果 A_i 来获得最终的比对结果 A 将是这种并行策略的关键技术.

2.2　C&E 方法

由于上述过程所产生的各中间结果是 S 序列的各子序列与 T 序列比对的结果, 所以既不能以获得分值最高的子序列比对结果做为最终结果, 也无法简单地对各分段比对结果进行拼接而合成最终结果. 我们提出了一种 C&E 方法对中间比对结果进行处理以得到最终比对结果, 其基本思想是, 首先对所有的中间比对结果按照其分值进行降序排列, 然后按顺序依次对中间结果以及其相邻片段的比对结果进行处理, 以得到最终的比对结果. 我们以 p=2 为例来详细说明中间结果的处理过程. 当 p=2 时中间比对结果 A_0 和 A_1 可以分为以下 6 种情况来考虑(如图 4 所示).

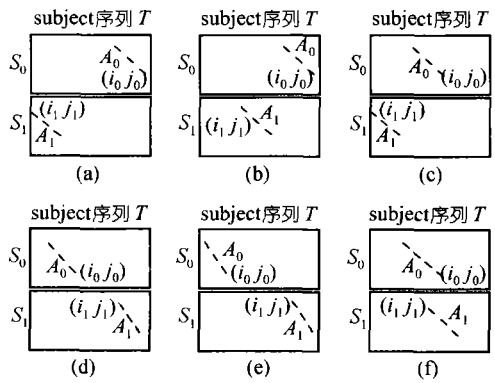

图 4　中间比对结果 A_0 和 A_1 的 6 种关系

图 4 中的方框表示动态规划矩阵 D_i, 方框中的虚线表示中间结果 A_i, 其所对应的分值为 M_i. 假设 A_0 的末端为 E, 对应于矩阵 D_0 中的 (i_0, j_0) 项, 表示比对 A_0 末端的字符是子序列 S_0 的第 i_0 个字符和序列 T 的第 j_0 个字符. A_1 的起始端为 H, 对应于矩阵 D_1 中的 (i_1, j_1) 项, 表示比对 A_1 的起始端的字符是子序列 S_1 的第 i_1 个字

符和序列 T 的第 j_1 个字符.

情况 a: 如图 4(a)所示, 可描述为比对 A_0 是由子序列 S_0 的一部分和序列 T 的末端部分共同构成的, 而比对 A_1 是由子序列 S_1 的一部分和序列 T 的起始端部分共同构成的, 在图 4(a)中 $j_0 = |T|, j_1 = 1$(如图 5 所示).

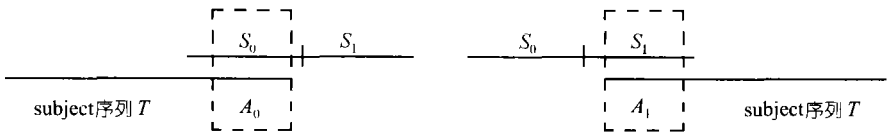

图 5 情况 a 示意图

由于 A_0 的末端 E 已经抵达序列 T 的末端, A_1 的起始端 H 已经抵达序列 T 的起始端, 即 A_0 无法向下延展, 而 A_1 也无法向上延展. 因此, 在这种情况下序列比对的最终结果 A 即为 $\max\{M_0, M_1\}$所对应的比对.

情况 b: 如图 4(b)所示, 可描述为 A_0 的末端 E 已经抵达序列 T 的末端, 而 A_1 的起始端 H 未到达序列 T 的起始端, 即 $j_0 = |T|, j_1 > 1$. 因此, A_0 无法向下延展, 而 A_1 却可以向上延展. A_1 向上延展的算法与动态规划回溯算法类似, 由于在延展过程中会涉及到不匹配或空位, 从而使比对的分值下降, 因此我们设置一个阈值参数 θ, 当比对分值低于阈值 θ 时延展终止. 如果 A_1 延展至矩阵 D_1 的顶端, 而此时 $j_1 > 1$, 即 A_1 未延展至序列 T 的起始端, 则将 A_1 的顶端信息传递给处理器 P_0, 并根据回溯算法, 从相应的位置继续对 A_1 延展. 假设 A_1 最终延展为 A'_1, 其对应的分值为 M'_1, 则必须满足 $M'_1 > M_1$. 序列比对的最终结果 A 即为 $\max\{M_0, M'_1\}$ 所对应的比对.

情况 c: 如图 4(c)所示, 此时 A_1 无法向上延展, 而 A_0 却可以向下延展. A_0 向下延展可采用逆回溯算法. 同样当比对分值低于阈值 θ 时延展终止. 当 A_0 延展至矩阵 D_0 的底端时, 需要将 A_0 的底端信息传递给处理器 P_1, 对 A_0 继续延展. 假设 A_0 最终延展为 A'_0, 其对应的分值为 M'_0, 则必须满足 $M'_0 > M_0$. 序列比对的最终结果 A 即为 $\max\{M'_0, M_1\}$所对应的比对.

对于情况 d 和 e, 我们令 $K = \max\{(|S_0| - i_0 + i_1), (j_1 - j_0)\}$, 令 $miscost$ 表示字符不匹配(包含添加空位)时的罚分.

情况 d: 如图 4(d)所示, 此时 $j_0 < |T|, j_1 > 1, j_0 < j_1$ 且 $K \times miscost < \min\{M_0, M_1\}$. 从图中可以看出子序列 S_0 和序列 T 的前半部分存在着匹配关系, 子序列 S_1 和序列 T 的后半部分存在着匹配关系, 即 A_0 和 A_1 不存在重叠的区域. 由于 $K \times miscost < \min\{M_0, M_1\}$, 使得$(M_0 + M_1 - K \times miscost) > \max\{M_0, M_1\}$, 因此在这种情况下只需将 A_0 和 A_1 通过一些不匹配的比对或添加一些空位连接起来, 即可获得序列 S 和 T 的局部最优比对 A. 在连接 A_0 和 A_1 时要尽量沿着对角线的方向进行连接, 以使得添加的不匹配字符或间隔的个数最少.

情况 e: 如图 4(e)所示, 此时 $j_0 < |T|, j_1 > 1, j_0 < j_1$ 但 $K \times miscost > \min\{M_0, M_1\}$.

虽然与情况 d 类似(A_0和A_1不存在重叠的区域), 但由于 $K\times miscost > \min\{M_0, M_1\}$, 如果按照情况 d 的处理方法, 将使得 $M_0 + M_1 - K\times miscost < \max\{M_0, M_1\}$. 该情况的处理方法是: 同时对 A_0 向下延展, 对 A_1 向上延展, 延展的方法与情况 b 和 c 中的方法相同. 延展的前提条件是: 延展后比对的分值必须大于延展前比对的分值. 假设 A_0 和 A_1 延展后的分值分别为 M'_0 和 M'_1, 则序列比对的最终结果 A 即为 $\max\{M'_0, M'_1\}$所对应的比对.

情况 f:　如图 4(f)所示, 此时 $j_0 < |T|, j_1 > 1$, 且 $j_0 > j_1$. 从图中可以看出, A_0 和 A_1 存在着重叠的区域, 即序列 T 中的某一部分同时与 S_0 和 S_1 中的一部分存在着比对关系. 该情况的处理方法与情况 e 相同.

当处理器的个数 $p > 2$ 时, 其中间比对结果 A_i 的处理方法描述如下:

1) 将所有的 A_i 按照其分值 M_i 的大小进行降序排列;

2) 取分值最大的两个比对, 分别表示为 A_α 和 A_β, 其所在的处理器分别为 P_α 和 P_β;

3) 如果 A_α 和 A_β 相邻, 按照 $p = 2$ 时的方法处理, 其结果记作 A_α;

4) 如果 A_α 和 A_β 不相邻, 则沿着 A_α 到 A_β 的方向, 对 A_α 和其相邻的中间结果按 p=2 时的方法处理, 其结果记作 A_α;

4.1) 此时如果 A_α 和 A_β 相邻, 则转 3),

4.2) 否则沿着 A_β 指向 A_α 的方向, 对 A_β 和其相邻的中间结果照 $p = 2$ 时的方法处理, 其结果记作 A_β;

5) 重复执行 2)~4)步, 对所有的中间结果 A_i 处理完毕后, 得到序列的最终结果 A.

3　实验结果

我们在曙光 2000- I 机群系统上采用 C 语言和 MPI 实现了上文提到的 PSW-DC 算法, 并且针对不同长度的生物序列, 使用不同规模的处理器进行了测试, 得到的数据如表 1 和 2.

表 1　不同序列在不同规模的处理器上的计算时间

序列长度	处理器个数				
$\|S\| = m, \|T\| = n$	1	2	4	8	16
m = 4k, n = 4k	3.0041	1.9319	0.975	0.512	0.267
m = 8k, n = 8k	15.761	10.315	4.9814	2.955	1.66
m = 16k, n = 16k	53.832	34.06	17.682	9.49	5.38

从表中的实验数据可以看出, 对于相同规模的处理器而言, 随着序列长度的增加相应的最终比对结果的灵敏度将会提高; 而对于相同的序列, 随着处理器个数的增加, 分配到每个处理器上的 query 子序列的长度将会减小, 使得最终比对

表 2 不同序列在不同规模的处理器上的灵敏度比较

处理器个数	序列长度, $\|S\| = m$, $\|T\| = n$		
	m = 1k, n = 1k	m = 2k, n = 2k	m = 4k, n = 4k
2	0.9436	0.9487	0.9631
4	0.9213	0.9207	0.9481
8	0.8367	0.8533	0.8812
16	0.6992	0.7153	0.7426

结果的灵敏度较低. 由于在生物学中, query 序列的种类和功能是通过其三维结构而不是序列比对的结果来判定的, 因此, 序列比对的结果仅仅是作为一种参考, 允许存在一定的误差. 由于生物学中复杂的机制, 关于序列比对允许的误差范围还没有一个公认的标准, 一种普遍默认的观点是只要算法能够在速度上有很大的提高, 相应牺牲一些结果的质量也是可取的. 从图 6 中可以看到, 随着处理器的增加, 计算速度会有很大的提高.

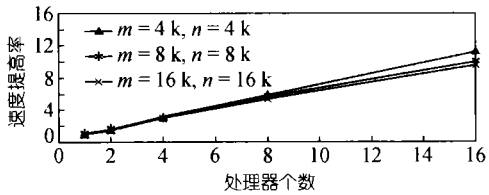

图 6 PSW-DC 算法对不同序列的加速比

4 结论

基于分而治之策略, 本文提出了一种新的并行生物序列比对算法, PSW-DC 算法. 根据我们的算法, 原始 query 序列按照处理器的个数划分为若干子序列并分配到相应的处理器上, 然后, 每个处理器独立地对所分配的数据运行 Smith-Waterman 算法, 并分别产生一个中间比对结果, 最后对各个处理器计算的中间结果进行处理, 得到最终最优比对结果. 同时我们给出了对中间比对结果进行处理以获得最终比对结果的处理方法, C&E 方法. 该算法已经在机群系统 DAWNING 2000-I 上对其获得了实现.

与其他的并行 Smith-Waterman 算法相比, 我们的算法有效地降低了对内存空间的需求, 原并行算法需要存储一个$(n+1)\times(m+1)$的矩阵, 如果该矩阵的规模超过了机群系统中每个处理器的内存空间, 算法将无法对数据进行处理, 而按照我们的算法, 每个处理器只需要存储原算法所需内存空间的 $1/p$(p 为处理器个数), 因此对于上述原算法无法处理的数据, 利用我们的 PSW-DC 算法可以较容易进行处理.

如何降低算法对内存的需求以及提高算法的运行速度已经成为生物信息学中的一个关键问题. 本文所提出的并行算法, 采用分而治之的策略, 有效地降低了算法对内存的需求, 并提高了算法的运行速度. 因此, 本文所提到的并行算法的基本思想, 可以应用于其他的序列比对或多条序列比对算法中.

本文提出的归并各分段比对中间结果而合成最终结果的过程是一个递归的

计算过程. 归并中的不同的优先选择过程和拓展规则均可能对于最终比对结果和生物序列比对过程的整体的计算量产生影响. 如何以尽量小的计算量和存储量(每个处理器内存空间的需求)生成尽量优化的最终比对结果, 是值得进一步研究的课题.

参　考　文　献

1 Aluru S, Futamura N, Mehrotra K. Parallel biological sequence comparison sing prefix computations. In: Proceedings of the 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing, 1998

2 Needleman S B, Wunsch C D. A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology, 1970, 48: 443~453

3 Smith T F, Waterman M S . Identification of common molecular subsequences. Journal of Molecular Biology, 1981, 147(1): 195~197

4 Altschul S F, Gish W, Miller W, et al. Basic local alignment search tool. Journal of Molecular Biology, 1990, 215:403~410

5 Altschul S F, Madden T L, Schaffer A A, et al. Gapped BLAST and PSI-BLAST: A new generation of protein database search program. Nucleic Acids Res, 1997, 25(17): 3389~3402

6 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

7 Lander E. Protein sequence comparison on a data parallel computer. In: Proceedings of the 1988 International Conference on Parallel Processing, 1988. 257~263

8 Galper A R, Brutlag D L. Parallel similarity search and alignment with the dynamic programming method. Technical Report, California: Stanford University, 1990

9 Sankoff D. The early introduction of dynamic programming into computational biology. Bioinformatics, 2000, 16(1): 41~47

10 Waterman M S. Introduction to Computational Biology, Maps, Sequences and Genomes. London: Chapman & Hall, 1998