愛知工業大学研究報告
第40号B 平成17年

計算クラスタ上で文字列の類似度を計算するための並列アルゴリズム

A Parallel Algorithm to Calculate the Similarity of Two Strings on a Cluster of Computers

鈴木　晋*　　水野　勝教*　　石井　直宏*
Susumu Suzuki*　　Katsunori Mizuno*　　Naohiro Ishii*

Abstract　We present a parallel algorithm to calculate the similarity of two strings on computers connected by a local area network, called a cluster of computers. Let the length of each string be n and the number of the computers p. We show that the parallel algorithm can solve the problem in time $O(\frac{n^2}{p})$ when $p \leq \sqrt{\frac{n}{2B}}$ $(\simeq \frac{\sqrt{n}}{15})$, and, therefore, that the algorithm can do it in time $O(\sqrt{B}n^{\frac{3}{2}})$ $(\simeq O(10n^{\frac{3}{2}}))$ especially when $p = \sqrt{\frac{n}{2B}}$, where B is (physical time for the computer to transmit one word through the network)/(physical time for the computer to execute one instruction on CPU) $(\simeq 100)$. Since the time complexity of the sequential algorithm is $O(n^2)$, the parallel algorithm is faster than the sequential algorithm.

1　はじめに

遺伝子工学の進展にともない大量の遺伝情報が蓄積されるようになり，それらを計算機を使って高速に処理することが急務となっている．DNA の塩基配列の類似度の計算やタンパク質のアミノ酸配列の類似度の計算はそのようなものの 1 つである．これらの計算は 2 つの文字列の類似度の計算と見なすことができる [1, 2].

ところで，計算機工学の世界では，1990 年代から多数のワークステーションやパソコンを LAN で接続したシステム上で並列処理を行う研究が盛んに行われるようになり，現在では，MPI (Message Passing Interface) や SCore 等の並列処理のための基盤ソフトが無償で提供されている．このように構成される並列計算システムは計算クラスタと呼ばれる [3, 4]. 計算クラスタを使って問題を高速に解くためには，問題を並列的に処理する並列アルゴリズムが必要である．特に計算クラスタでは B =(1 語の転送時間)/(1 命令あたりの CPU 時間) がかなり大きい（約 100）ため，通信時間が爆発しないように適切な粒度で問題を並列処理することができる並列アルゴリズムが重要である．

本稿では，文字列類似度問題を計算クラスタを使って高速に解くことを目的として並列アルゴリズムを提案し，その速さを評価する．文字列の長さを n，計算機の台数を p とする．提案する並列アルゴリズムが，(1) $p \leq \sqrt{\frac{n}{2B}}$ $(\simeq \frac{\sqrt{n}}{15})$ のとき，時間量 $O(\frac{n^2}{p})$ で問題を解けること，故に特に，(2) $p = \sqrt{\frac{n}{2B}}$ のとき，時間量 $O(\sqrt{B}n^{\frac{3}{2}})$ $(\simeq O(10n^{\frac{3}{2}}))$ で問題を解けることを示す．1 台の計算機を使って問題を解く逐次アルゴリズムの時間量は $O(n^2)$ であるので，(1) より，計算機が $\frac{\sqrt{n}}{15}$ 台以下のときは，並列アルゴリズムは逐次アルゴリズムより高速であり，その速さは使用できる計算機の台数に比例して速くなる．また，(2) より，$\frac{\sqrt{n}}{15}$ 台程度の計算機が使用できるときは，並列アルゴリズムは逐次アルゴリズムより大幅に高速である．

2　文字列類似度問題

2 つの文字列，例えば，"*east*" と "*west*" をギャップを挿入しながら並べたときに，同じ位置にある 2 つの文字からなる文字対で両文字が一致するような文字対（一致文字対と呼ぶ）を最大何個作れるかを考える．この例の文字列の場合，`-east` / `we-st` のようにギャップ ('-' と記す) を挿入して並べると，全文字対 ('-','w'), ('e','e'), ('a','-'), ('s','s'), ('t','t') のうち，3 つの文字対 ('e','e'), ('s','s'), ('t','t') が一致文字対になる．また，どのようにギャップを挿入しても，一致文字対を 3 つより多く作ることはできない．故に，"*east*" と "*west*" に対しては，一致文字対の最大数は 3 である．

* 愛知工業大学 経営情報科学部 情報科学科 コンピュータシステム専攻（豊田市）

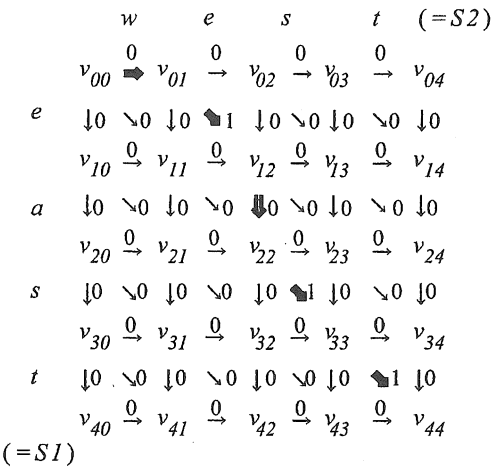

図 1. コスト付きグラフ G

文字列類似度問題は，上の例のように，2 つ文字列が与えられて，ギャップを挿入してそれらの文字列を並べたときの，ある目標関数 f（上の例では一致文字対の数）の最大値（上の例では 3），および，その値を与える文字列の並べ方（上の例では $\begin{matrix}\texttt{-east}\\ \texttt{we-st}\end{matrix}$）を求める問題である．最大値を文字列の類似度，文字列の並べ方をアライメント，特に，最大値を与える文字列の並べ方を最適アライメントという．目標関数 f は，上の例では $f = 1 \times$ (一致文字対の個数) であったが，一般には

$$\begin{aligned} f = {} & c_I \times (\text{一致文字対の個数}) \\ & + c_H \times (\text{不一致文字対の個数}) \\ & + c_K \times (\text{ギャップ文字対の個数}) \end{aligned}$$

と表される．ここで，c_I, c_H, c_K は応用に適したある重みであり，不一致文字対は ('a','b') のように異なる文字からなる対，ギャップ文字対は ('a','-') のように文字とギャップからなる対である．

本稿では，説明の簡単のため，文字列の類似度のみを求める（最適アライメントを求めるように拡張することは容易である）．また，2 つの文字列の長さは等しいとする（1 節で述べたように n と記す）．

3 逐次アルゴリズムの紹介

文字列類似度問題はグラフ上の経路探索問題として解くことができる [1, 2]．2 節の問題例（文字列が $S1 =$ "$east$" と $S2 =$ "$west$"，目標関数 f の中の係数が $c_I = 1, c_H = 0, c_K = 0$）を使って説明する．問題例に対して図 1 のコスト付きグラフ G を考える．G では，$(|S1|+1)(|S2|+1) = (4+1)(4+1)$ 個の節点 $v_{ij}, i, j = 0, \ldots, 4$, が格子状に並んでおり，横方向，縦方向，斜め方向に規則正しく枝が出ている．文字列 S の i 番目の文字を $S(i)$ と記す．G の各枝にはコストがついており，横方向および縦方向の枝のコストは $c_K = 0$，斜め方向の枝 $(v_{ij}, v_{i+1,j+1})$ のコストは，$S1(i+1) = S2(j+1)$ のとき $c_I = 1$，$S1(i+1) \neq S2(j+1)$ のとき $c_H = 0$ である．G 上の路のコストを路に現れる枝のコストの和と定義する．例えば，図 1 の太い枝からなる路 $v_{00}v_{01}v_{12}v_{22}v_{33}v_{44}$ のコストは $0+1+0+1+1 = 3$ である．このように G および路のコストを定めると，G の左上の節点 v_{00} から右下の節点 v_{44} へ至るすべての路のコストの中の最大値が文字列 $S1 =$ "$east$" と $S2 =$ "$west$" の類似度になる．最大値を与える路を最適路と呼ぶ．路 $v_{00}v_{01}v_{12}v_{22}v_{33}v_{44}$ は最適路になっており，この路のコスト 3 が文字列の類似度になる（2 節で述べた類似度 3 に一致する）．

G の最適路のコスト（文字列の類似度）の計算は図 2 の動的計画法により行うことができる．ここで，w_{ij} は斜め方向の枝 $(v_{i-1,j-1}, v_{ij})$ のコスト，D_{ij} は節点 v_{00} から節点 v_{ij} への最適路のコストであり，D_{nn} が求める最適路のコスト（文字列の類似度）である．動的計画法の時間量は $O(n^2)$ である．

$$\begin{aligned} & D_{00} = 0 \\ & D_{0j} = D_{0,j-1} + c_K, \quad 1 \le j \le n \\ & D_{i0} = D_{i-1,0} + c_K, \quad 1 \le i \le n \\ & D_{ij} = \max\{D_{i-1,j} + c_K, D_{i-1,j-1} + w_{ij}, D_{i,j-1} + c_K\}, \\ & \qquad 1 \le i, j \le n. \end{aligned}$$

図 2. 動的計画法

4 並列アルゴリズム

計算機クラスタは 1 台のマスタ計算機と p 台のスレーブ計算機（スレーブ#1,..., スレーブ#p と記す）をバス型 LAN で接続して構成される．ただし，マスタの仕事は少ないため，マスタとスレーブ#1 は物理的に同一の計算機とする．送信，受信とも，通信はすべて同期型とする．D_{nn}（文字列の類似度）を計算する並列アルゴリズムを次に示す．

[マスタ計算機]

step1: すべてのスレーブに文字列データを送信する．
step2: スレーブ#p から D_{nn}（文字列の類似度）を受信したら，それを表示して終了する．□

[スレーブ計算機] /* スレーブ#k とする．*/

step1: マスタから文字列データを受信する．
step2: $k = 1$ ならば step3 へ，$2 \le k \le p-1$ ならば step4 へ，$k = p$ ならば step5 へ．

step3: /* スレーブ#1における処理 */
for($i=1; i\le p; p++$){
B_{1i}を計算する.
C_{1i}をスレーブ#2に送信する.
}
終了する.
step4: /* スレーブ#k ($2\le k\le p-1$) における処理 */
for($i=1; i\le p; p++$){
スレーブ#$(k-1)$から$C_{k-1,i}$を受信する.
B_{ki}を計算する.
C_{ki}をスレーブ#$(k+1)$に送信する.
}
終了する.
step5: /* スレーブ#pにおける処理 */
for($i=1; i\le p; p++$){
スレーブ#$(p-1)$から$C_{p-1,i}$を受信する.
B_{pi}を計算する.
}
D_{nn}をマスターに送信する.
終了する. □

ここで, B_{ki}, C_{ki}は次式で表される$D_{i'j'}$の集合である.

$$B_{ki}=\Bigg\{D_{i'j'}\mid \left\lceil\frac{n+1}{p}\right\rceil(k-1)\le i'\le\min\left\{\left\lceil\frac{n+1}{p}\right\rceil k-1,n\right\},$$
$$\left\lceil\frac{n+1}{p}\right\rceil(i-1)\le j'\le\min\left\{\left\lceil\frac{n+1}{p}\right\rceil i-1,n\right\}\Bigg\},$$
$$1\le k,i\le p. \qquad (1)$$

$$C_{ki}=\Bigg\{D_{i'j'}\mid i'=\left\lceil\frac{n+1}{p}\right\rceil k-1,$$
$$\max\left\{0,\left\lceil\frac{n+1}{p}\right\rceil(i-1)-1\right\}\le j'\le\left\lceil\frac{n+1}{p}\right\rceil i-1\Bigg\},$$
$$1\le k\le p-1,\ \ 1\le i\le p. \qquad (2)$$

並列処理の流れを$p=3, n=8$の例を使って説明する. 図3に示すように, $(n+1)\times(n+1)=9\times 9$の行列として表された全ての$D_{i'j'}$を$p\times p=3\times 3$個のブロックに分け, $B_{11},\ldots,B_{33}$と記す(式(1)のB_{ki}). また, B_{ki}と$B_{k+1,i}$の境界付近にある$D_{i'j'}$を$C_{11},\ldots,C_{23}$と記す(式(2)のC_{ki}). 初め, スレーブ#1は, (1.1) B_{11}(に含まれる$D_{i'j'}$)を図2と類似の動的計画法を使って計算し, 計算結果C_{11}(に含まれる$D_{i'j'}$)をスレーブ#2に送信する. その後, (1.2) B_{12}を計算し, C_{12}をスレーブ#2に送信し, (1.3) B_{13}を計算し, C_{13}をスレーブ#2に送信する. スレーブ#2は, (2.1) C_{11}を受信すると, B_{21}を計算し, C_{21}をスレーブ#3に送信する. その後, (2.2) C_{12}を受信するとB_{22}を計算し, C_{22}をスレーブ#3に送信し, その後, (2.3) C_{13}を受信するとB_{23}を計算し, C_{23}をスレーブ#3に送信する. スレーブ#3は, (3.1) C_{21}を受信するとB_{31}を計算し, その後, (3.2) C_{22}を受信するとB_{32}を計算し, その後, (3.3) C_{23}を受信するとB_{33}を計算する. 解D_{88}が得られたので, D_{88}をマスタに送信する. 表1は各時刻(段階)におけるスレーブの処理内容を示す. 第1段階ではスレーブ#1のみが処理(B_{11}の計算とC_{11}の送信)を行うが, 例えば, 第3段階では全てのスレーブが処理を行う. 最後に, 第5段階でスレーブ#3がB_{33}を計算して, 並列処理が終了する.

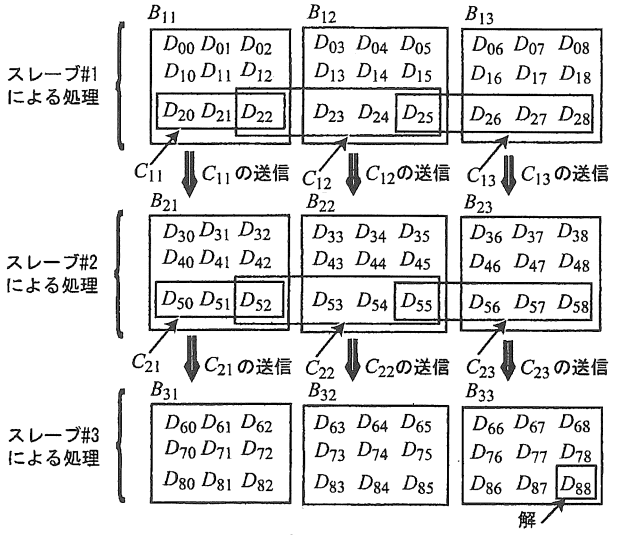

図3. 並列処理の流れ

表1. 各時刻(段階)におけるスレーブの処理内容

	第1段階	第2段階	第3段階
スレーブ#1	B_{11}の計算 C_{11}の送信	B_{12}の計算 C_{12}の送信	B_{13}の計算 C_{13}の送信
	第2段階	第3段階	第4段階
スレーブ#2	B_{21}の計算 C_{21}の送信	B_{22}の計算 C_{22}の送信	B_{23}の計算 C_{23}の送信
	第3段階	第4段階	第5段階
スレーブ#3	B_{31}の計算	B_{32}の計算	B_{33}の計算

5 計算時間の評価

各C_{ki}の転送時間は$A'+B'(\lceil\frac{n+1}{p}\rceil+1)$と近似することができる. ここで, A'は通信起動時間, B'は1語当たりの転送時間である[3]. また, Cを1命令当たりのCPU時間とすると, 各B_{ki}の計算時間はおよそ$C\lceil\frac{n+1}{p}\rceil^2$である. 並列処理は$(2p-1)$個の段階よりなり, 各段階において, 各スレーブのB_{ki}の計算は並行して行われ, 一方, 全てのC_{ki}の転送は逐次的に行わ

れる．故に，並列処理にかかるすべての時間 $T_t(p)$ は次式で与えられる：

$$T_t(p) \simeq (2p-1)\left(C\left\lceil\frac{n+1}{p}\right\rceil^2 + p\left(A' + B'\left(\left\lceil\frac{n+1}{p}\right\rceil+1\right)\right)\right) \simeq 2C\left(\frac{n^2}{p} + Ap^2 + Bnp\right), \quad (3)$$

ここで，$A = A'/C, B = B'/C$．時間 $T_t(p)$ を CPU のステップ数に換算したもの，すなわち，時間量を $T(p)$ と記す．(3) より，

$$T(p) = O\left(\frac{n^2}{p} + Ap^2 + Bnp\right). \quad (4)$$

以下，時間量 $T(p)$ を評価する．

式 (4) の括弧内を $g(p)$ と記す：

$$T(p) = O(g(p)), \quad (5)$$

$$g(p) = \frac{n^2}{p} + Ap^2 + Bnp. \quad (6)$$

このとき，

$$g'(p) = -\frac{n^2}{p^2} + 2Ap + Bn. \quad (7)$$

$g'(p) = 0$ の正の解を p_o と記すと

$$-\frac{n^2}{p_o^2} + 2Ap_o + Bn = 0, \quad (8)$$

$g'(p)$ の符号より，$g(p)$ は $0 < p \leq p_o$ のとき単調に減少し，$p = p_o$ のとき最小になる．$p \leq p_o$ とする．(8) より

$$\frac{n^2}{p_o} = p_o(2Ap_o + Bn) > p_o(Ap_o + Bn). \quad (9)$$

また，$p \leq p_o$ より

$$\frac{n^2}{p} \geq \frac{n^2}{p_o}, \quad Ap^2 + Bnp \leq Ap_o^2 + Bnp_o. \quad (10)$$

(6), (9), (10) より，

$$g(p) \leq \frac{n^2}{p} + Ap_o^2 + Bnp_o < \frac{n^2}{p} + \frac{n^2}{p_o} < \frac{2n^2}{p}. \quad (11)$$

(5), (11) より，$p \leq p_o$ のとき，

$$T(p) = O(g(p)) = O\left(\frac{n^2}{p}\right). \quad (12)$$

p_o より小さい p_o の近似値を求める．$-\frac{n^2}{p^2} + 2Bn = 0$ の解を p_a と記す：

$$-\frac{n^2}{p_a^2} + 2Bn = 0, \quad (13)$$

すなわち

$$p_a = \sqrt{\frac{n}{2B}}.$$

もし，n が

$$(p_a =)\sqrt{\frac{n}{2B}} \leq \frac{Bn}{2A} \quad (14)$$

すなわち

$$n \geq \frac{2A^2}{B^3} \quad (15)$$

を満たすならば，(7)，(13)，(14) より

$$g'(p_a) = -\frac{n^2}{p_a^2} + 2Ap_a + Bn \leq -\frac{n^2}{p_a^2} + 2Bn = 0$$

であるので，$g'(p)$ の単調増加性より

$$p_a \leq p_o.$$

A, B は各々 $10^4, 10^2$ 程度であるので，(15) は $n \geq 1000$ ($\gg 200$) であれば成立する．故に，並列計算の対象となるような大きさの n に対しては，$p_a = \sqrt{\frac{n}{2B}} \simeq \frac{\sqrt{n}}{15}$ を p_o より小さい p_o の近似値として使うことができる．

(12) より，$p \lesssim \frac{\sqrt{n}}{15}$ ($= p_a$) のとき，並列アルゴリズムは逐次アルゴリズム（時間量 $O(n^2)$）より高速であり，その速さは計算機の数 p に比例して速くなる．また，

$$T(p_a) = T\left(\sqrt{\frac{n}{2B}}\right) = O\left(\sqrt{2B}n^{\frac{3}{2}}\right) = O\left(\sqrt{B}n^{\frac{3}{2}}\right) \simeq O(10n^{\frac{3}{2}})$$

であるので，$p \simeq \frac{\sqrt{n}}{15}$ のとき，並列アルゴリズムは逐次アルゴリズムより大幅に高速である．

謝辞 この研究は愛知工業大学教育・研究特別助成の援助を受けて行われました．

参考文献

[1] 中村春木，中井謙太: バイオテクノロジーのためのコンピュータ入門，コロナ社，東京，1995.

[2] Dan Gusfield: Algorithms on Strings, Trees, and Sequences – Computer Science and Computational Biology –, Cambridge University Press, New York, 1997.

[3] Barry Wilkinson and Michael Allen: Parallel Prongramming – Techniques and Applications Using Networked Workstations and Parallel Computers –, Prentice-Hall, 1999.

[4] 石川裕，佐藤三久他: Linux で並列処理をしよう，共立出版，東京，2002.

（受理　平成 17 年 3 月 17 日）