Vol.2014-BIO-40 No.3
2014/12/18

塩基配列の全対比較のための枝刈りによるSmith-Waterman アルゴリズムの高速化

岡田 大輝[1]　伊野 文彦[1]　萩原 兼一[1]

概要：塩基配列の全対比較とは，複数本の配列に対し，それらの重複なしの組み合わせに関してアライメントを出力することである．本論文では，塩基配列に対する全対比較の高速化を目的として，SW（Smith-Waterman）アルゴリズムのための枝刈り手法を提案する．提案手法は，計算済みの配列対のスコアを基に，残りの対におけるスコアの下界を導出し，効率のよい枝刈りを実現する．提案手法を GPU（Graphics Processing Unit）上で評価した結果，対ごとに枝刈りする既存手法と比べて最大で 1.2 倍の速度向上を得た．

キーワード：ローカルアライメント，Smith-Waterman アルゴリズム，全対比較，枝刈り

Accelerating the Smith-Waterman Algorithm with a Pruning Method for All-Pairs Comparison of Base Sequences

Daiki Okada[1]　Fumihiko Ino[1]　Kenichi Hagihara[1]

Abstract: All-Pairs comparison of base sequences outputs alignments for combinations of multiple sequences without repetition. In this paper, we propose a pruning method for the Smith-Waterman algorithm, aiming at accelerating all-pairs comparison of base sequences. Our method realizes efficient pruning by deriving a lower bound for unprocessed pair of sequences from a score computed for a pair of sequences. Experimental results on a graphics processing unit (GPU) show that our method is 1.2 times faster than a previous method that performs pairwise pruning.

Keywords: local alignment, Smith-Waterman algorithm, all-pairs comparison, pruning

1. はじめに

ペアワイズアライメントとは，塩基やアミノ酸などの生体配列の対に対し，それらの類似部位を特定する操作である．この操作は，分子系統樹の作成，遺伝子情報の解析，およびタンパク質の構造解析などに用いられる．特に，局所的な類似部位を特定する処理をローカルアライメントと呼ぶ．ローカルアライメントは，遺伝子の置換や欠損のある部位を特定できるため，生物の進化を表す分子系統樹を作成する際に有用である．

ローカルアライメントの厳密解，すなわち配列間の類似性を表すスコアおよび類似部位を出力するアルゴリズムとして，動的計画法に基づく SW（Smith-Waterman）アルゴリズム [1] が知られている．比較対象となる配列の長さを m および n $(\geq m)$ とすると，その時間計算量は $\mathcal{O}(mn)$ である．塩基配列の長さは数十億塩基対（BP，Base Pair）に達するため，その計算時間の短縮を目的として，FPGA（Field Programmable Gate Array）[2] や Xeon Phi [3] などの様々なハードウェアを用いた高速化が試みられている．なかでも，グラフィクス処理を加速する GPU（Graphics Processing Unit）[4] は，有望な汎用アクセラレータとして活用されている．

CUDAlign 1.0 [5] は，SW アルゴリズムの一部を単一

[1] 大阪大学大学院情報科学研究科コンピュータサイエンス専攻
Department of Computer Science, Graduate School of Information Science and Technology, Osaka University

GPU 上で並列処理するものであり，ヒトの 21 番染色体とチンパンジーの 22 番染色体に対するローカルアライメントのスコアを初めて計算した．これらの配列長はそれぞれ 47MBP および 33MBP であり，GeForce GTX 280 上で 21 時間を要した．また，このスコアを基に類似部位を特定し，枝刈りにより SW アルゴリズムの高速化を実現した [6]．さらに，64 台の Tesla M2090 を用い，ヒトとチンパンジーの 1 番染色体（長さ 249MBP および 228MBP）に対するローカルアライメントを 9 時間で得た [7]．SW# [8] は，2 台の GPU を用いて SW アルゴリズムの並列処理を実現した．この実装は，GeForce GTX 690 を用いて，ヒトの 21 番染色体とチンパンジーの 22 番染色体に対するローカルアライメントを 6 時間半で処理した．

しかし，これらの既存研究はいずれも 1 対の配列を高速化の対象としている．正確な系統樹を作成するためには，対象とする生体配列の全対比較が必要であり [9]，全対比較を含めた高速化によりさらなる時間短縮を果たせる可能性がある．ここで，全対比較とは，複数本の配列に対してそれらの重複なしの組み合わせに関してアライメントを施すことである．入力となる配列の数を N とすれば，$N \times (N-1)/2$ $(= {}_NC_2)$ 個のアライメント処理を必要とするため，ペアワイズアライメントのさらなる時間短縮が必要である．

そこで本論文では，全対比較の高速化を目的として，枝刈りにより SW アルゴリズムの実行時間を短縮する手法を提案する．提案手法は，計算済みの対のスコアを基に，残りの対におけるスコアの下界を導出し，効率のよい枝刈りを実現する．したがって，配列間の類似性が高い場合，高速化が期待できる．提案手法は，SW#のマルチ GPU 実装 [8] を基に，マルチ GPU 上で動作する．

以降では，2 節で関連研究を紹介し，3 節で SW アルゴリズムおよび既存の枝刈り手法 [6] についてまとめる．4 節で提案手法を説明し，5 節で実験結果を示す．最後に，6 節で本論文をまとめる．

2. 関連研究

Feng ら [9] は，複数本の配列から系統樹を作成する累進法を提案した．この手法は，総当りのペアワイズアライメントにより配列間の距離行列を計算し，系統樹を作成する．総当りの部分は逐次処理されていて，対間の枝刈りは実現していない．提案手法は，枝刈りにより総当りの高速化を図る．

累進法は，ClustalW [10] や T-Coffee [11] において採用されている．前者は，総当りのペアワイズアライメントを高速化するために，Wilbur ら [12] のアライメントアルゴリズムを採用している．このアルゴリズムは，一連の塩基対からなるフラグメントを定義し，フラグメント単位でアライメントを得る．例えば，長さが k のフラグメントを用い，長さ k 以上の一致を含む部位を基に，全体のアライメントを構築することで時間短縮を図る．後者も同様のアルゴリズムを採用していて，いずれも対間の枝刈りを考慮していない．

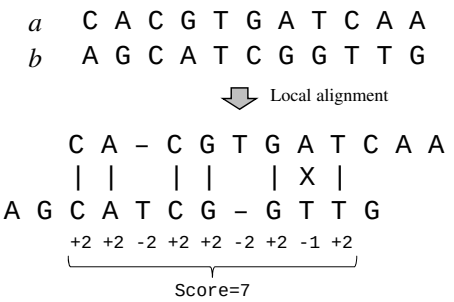


図 1 ローカルアライメントの例

同一対内の枝刈りは，CUDAlign 2.1 [6] が実現した．この枝刈り手法は，スコア計算時に下界を超えないことが確定した部分の計算を省略し，SW アルゴリズムの実行時間を短縮する．下界として，その時点で得られているスコアの最大値を用いる．これにより最大で 53.7%の計算を省略できている．しかし，枝刈りに用いる下界を対内の計算から得ているため，さらなる省略は難しいことが証明されている [6]．この枝刈り手法は，SW# [8] においても採用されている．

3. Smith-Waterman アルゴリズムと枝刈り

長さがそれぞれ m および n $(\geq m)$ の配列 a および b が与えられたとする．一般に，アライメントアルゴリズムは，文字の置換や挿入により a を b に変換する際の編集コストを基に，a および b の類似度（スコア）を計算する．ここで，遺伝子の置換は文字の置換に対応し，欠損はギャップ（空白）の挿入に対応する．互いに類似している配列は高いスコアを持つ（**図 1**）．

配列 a における i 番目の文字を a_i で表す．また，文字 a_i および b_j の類似度を関数 $s(i,j)$ で表し，以下で与える．

$$s(i,j) = \begin{cases} p, & \text{if } a_i = b_j, \\ q, & \text{otherwise.} \end{cases} \quad (1)$$

ここで，p (≥ 0) は文字 a_i および b_j が一致する場合のスコアを表し，q $(< p)$ はそれらが一致しない場合のスコアを表す．また，アフィンギャップペナルティを用い，長さ l のギャップのコストを $o + e \times (l-1)$ とする．ここで，o はギャップ開始ペナルティであり，e はギャップ延長ペナルティである．

SW アルゴリズム [1] は，行列計算およびトレースバックで構成されている（**図 2**）．前者は，任意の位置で終端するアライメントの最大スコアを動的計画法により計算する．後者は，それらの最大スコアを返す部分配列，すなわち解となる類似部位を，必要とする文字の置換や挿入とと

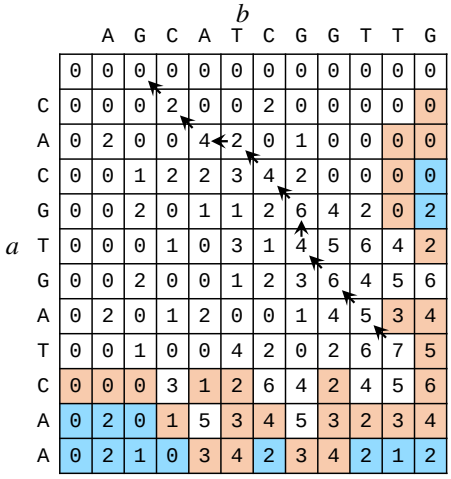


図 2 Smith-Waterman アルゴリズムにおける行列計算およびトレースバック

もに特定する．行列計算およびトレースバックはそれぞれ $\mathcal{O}(mn)$ 時間および $\mathcal{O}(m+n)$ 時間を必要とするため，前者が実行時間の大半を占める．

文字 a_i および b_j で終端する部分配列における最大スコアを $H_{i,j}$ $(0 \le i \le m, 0 \le j \le n)$ とする．このとき，$H_{i,j}$ は以下の漸化式で表せる [13]．

$$H_{i,j} = \max\{0, E_{i,j}, F_{i,j}, H_{i-1,j-1} + s(i,j)\} \quad (2)$$

$$E_{i,j} = \max\{E_{i,j-1} - e, H_{i,j-1} - o\} \quad (3)$$

$$F_{i,j} = \max\{F_{i-1,j} - e, H_{i-1,j} - o\} \quad (4)$$

ただし，任意の i に対して $H_{i,0} = E_{i,0} = F_{i,0} = 0$ であり，任意の j に対して $H_{0,j} = E_{0,j} = F_{0,j} = 0$ である．式 (2)〜(4) から，ある要素の計算は，その左上，上および左の要素を必要とする．

行列内のすべての要素を埋めたのち，得られたスコア，すなわち行列内の最大値を基点としてトレースバックが始まる．その最大値に到るまでの計算の過程を逆に辿ることにより，どの文字を置換すべきか，あるいはどこへ空白を挿入すべきかが分かる．図 1 の例では，配列 a における `CA-CGTGAT` および配列 b における `CATCG-GTT` がローカルアライメントの解である．ここで，文字「-」は空白を表す．

以降では，元の配列において解に対応する部分配列を類似部位と呼ぶ．配列 a における類似部位 x〜y は，x 番目から y 番目までの文字が配列 b と類似していることを表す．例えば，**図 4** の例では，$x = 1$ および $y = 8$ である．ここで，類似部位は文字の置換や空白を含まないことに注意されたい．同様に，配列 b における類似部位を u〜v $(1 \le u \le v \le |b|)$ と表す．

3.1 同一対内における枝刈り

CUDAlign 2.1 [6] の枝刈り手法は，SW アルゴリズムの行列計算を対象としている．この枝刈り手法は，$i \ge \lceil m/2 \rceil$ もしくは $i \ge n - j$ を満たす要素 $H_{i,j}$ に対する計算を省ける可能性がある [6]．計算を省略できる要素の数は，$m \le 2n$ のとき高々 $\lfloor mn/2 - m^2/8 \rfloor$ 個であり，$m > 2n$ のとき高々 $\lfloor mn - n^2/2 \rfloor$ 個である．

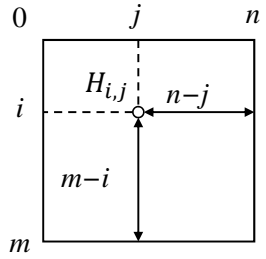


図 3 要素 $H_{i,j}$ の位置関係

枝刈り条件を示すための定義を示す．

定義 1. **$H_{i,j} + p \times \max(m-i, n-j) < L$ を満たす要素を起点要素と呼ぶ．ここで，L はスコアの下界を表し，$H_{i,j}$ を計算するまでに得られたスコアの最大値を下界として用いる．初期値は $L = 0$ である．**

定義 1 は，文字 a_i もしくは b_j 以降のすべての文字が一致したとしても，要素 $H_{i,j}$ から派生しうるすべてのスコアが最適値にならないことを調べている．**図 3** に示すように，以降の文字は高々 $\max(m-i, n-j)$ 個であるため，それらがすべて一致したときのスコア $H_{i,j} + p \times \max(m-i, n-j)$ を，$H_{i,j}$ から派生しうるスコアの最大値としている．

式 (2) より，要素 $H_{i,j}$ を枝刈りするための条件は以下のように導ける．

条件 要素 $H_{i-1,j-1}$，$H_{i-1,j}$ および $H_{i,j-1}$ が枝刈りされている，もしくは起点要素であること．

図 2 の例では，橙色の要素が起点要素であり，青色の要素が枝刈りできる要素である．

3.2 全対比較

比較対象とする配列の集合を $\mathcal{N} \triangleq \{1, 2, \cdots, N\}$ とする．また，$a > b$ を満たす配列対 $\langle a, b \rangle$ からなる集合を $\mathcal{M} \triangleq \{\langle a, b \rangle \mid a, b \in \mathcal{N}, a > b\}$ とする．全対比較とは，$\forall i \in \mathcal{M}$ に対し，そのスコア S_i とその類似部位 x〜y $(1 \le x \le y \le |a|)$ を計算することである．ここで，$|a|$ は配列 a の長さを表す．

4. 提案手法

対 $i = \langle a, b \rangle$ における類似部位 x_i〜y_i，文字の不一致数 m_i およびギャップ数 g_i が計算済みとする．対 $j = \langle a, c \rangle$ $(b \ne c)$ に対する x_j〜y_j，m_j および g_j についても同様である．これらの情報を基に，提案手法は対 $k = \langle b, c \rangle$ に対する行列計算を，効率のよい枝刈りにより高速化する．以降では，配列 a のうち，対 i および対 j の類似部位が共に含む部分配列を共通部と呼ぶ（図 4）．

定理 1. **対 i および対 j 間に共通部が存在しないための必要十分条件は $y_i < x_j$ もしくは $y_j < x_i$ である．すなわ**

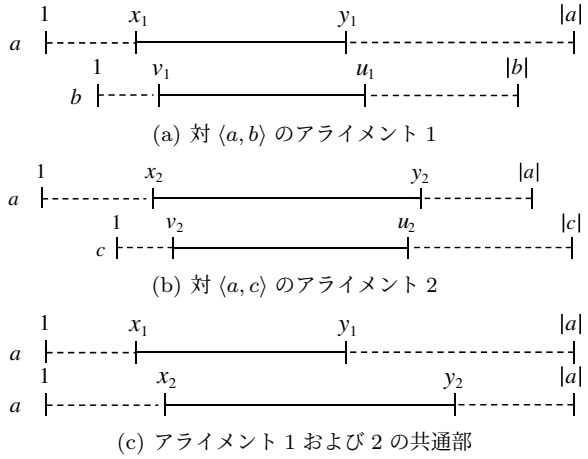


図 4 アライメントの共通部

ち，共通部が存在するための必要十分条件は $y_i \geq x_j$ かつ $y_j \geq x_i$ である．この条件を満たすとき，配列 a における共通部は $\max(x_i, x_j)$〜$\min(y_i, y_j)$ である．

4.1 スコアの下界

提案手法は，共通部 $\max(x_i, x_j)$〜$\min(y_i, y_j)$ におけるスコアの下界を対 k におけるスコアの下界として用いる．共通部において，対 i および j のすべて（$m_i + m_j$ 個）の不一致およびすべて（$g_i + g_j$ 個）のギャップが出現するとき，共通部のスコアが最小となる．このとき，共通部において一致する文字数 P は，共通部の長さ $\min(y_i, y_j) - \max(x_i, x_j) + 1$ からこれらを減算したものである．一方，不一致の総コスト Q は $q \times (m_i + m_j)$ であり，ギャップの総コスト G は $\max(o + e \times (g_i + g_j - 1), o \times (g_i + g_j))$ である．以上から対 k におけるスコアの下界 L は，式 (5) で表せる．

$$L = p \times P - Q - G \tag{5}$$

提案手法は，スコア計算の開始時に L を式 (5) で初期化しておく．それ以降のアルゴリズムの動作は既存手法 [6] と同一である．

L が下界であることを示す．仮に，対 k に対してスコアが L' $(< L)$ となるようなアライメントが存在すると仮定する．このとき，共通部におけるスコアの下界は L であるため，共通部を除く，残りの部分配列においてスコアは負の値でなければならない．しかし，すべての置換および空白の挿入が共通部に出現しているので，残りの部分配列ではすべての文字が一致している．一致しているときのコストは $p \geq 0$ であるため，矛盾．したがって，そのような L' は存在しない．

5. 評価実験

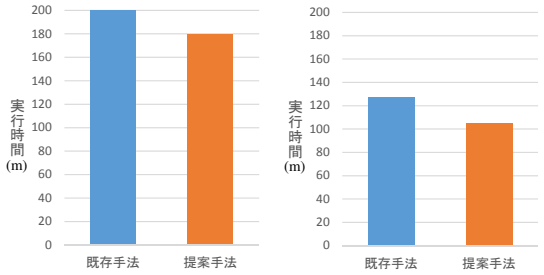
提案手法の性能を評価するために，実行時間や枝刈り率に関して既存手法 [6] と提案手法を比較した．提案手法は SW#を拡張することにより実装した．一方，既存手法の

表 1 実験環境

項目	仕様
CPU	Intel Xeon CPU E5-2680 v2 @ 2.80 GHz
GPU	Nvidia Kepler K40 ×2
RAM	512 GB
OS	Ubuntu 14.04.1
CUDA	CUDA 6.5

表 2 実験に用いた生体配列

配列	アクセッション番号	配列長（BP）	備考
1	AE016879.1	5,227,293	炭疽菌
2	CP002091.1	5,218,947	炭疽菌
3	CP001598.1	6,871,806	炭疽菌
4	CP001970.1	7,011,976	炭疽菌
5	AE017225.1	7,013,956	炭疽菌
6	CP001215.1	7,015,631	炭疽菌
7	NC_000019.10	58,617,616	ヒト
8	FR853090.1	56,181,278	ゴリラ
9	NC_006486.3	63,644,993	チンパンジー



(a) 1 GPU　(b) 2 GPU

図 5 全対比較に要する実行時間の比較

マルチ GPU 実装として，SW# [8] を用いた．

表 1 に，実験に使用した環境を示す．実験に用いた PC は，GPU を 2 枚装備している．スコアリング関数としては，既存研究 [6,8] と同一の $(p, q, o, e) = (1, -3, 5, 2)$ を用いた．

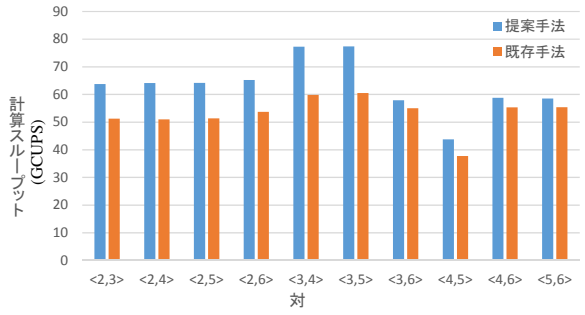
表 2 に，実験に用いた生体配列 [14] の一覧を示す．配列 1〜6 は炭疽菌の塩基配列である．配列 7，8 および 9 はそれぞれヒト，チンパンジーおよびゴリラの塩基配列であり，いずれも 19 番染色体のものである．

5.1 計算時間の比較

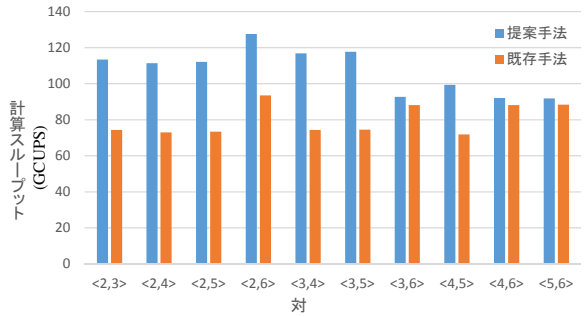
図 5 に，$\mathcal{N} = \{1, 2, 3, 4, 5, 6\}$ に対する全対比較に要した実行時間 T を示す．ここで，T はトレースバックが要した時間を含む．計測時には，最初に対 $\langle 1, *\rangle$ を解いた後，順に $\langle 2, *\rangle$，$\langle 3, *\rangle$，$\langle 4, *\rangle$ および $\langle 5, *\rangle$ を解いた．

提案手法は，1GPU 版の実行時間を 200 分から 180 分に短縮し，2GPU 版を 127 分から 105 分に短縮した．このときの速度向上率はそれぞれ 1.1 倍および 1.2 倍であり，提案手法の高速化効果は 2GPU 版の方が高かった．

対ごとの内訳を調べるために，両手法の計算スループット $\rho = (m + 1) \times (n + 1)/T$ を計測した（**図 6**）．計算ス



(a) 1 GPU



(b) 2 GPU

図 6 計算スループットの比較

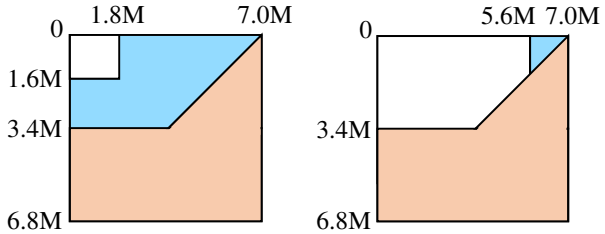
ループットの単位は CUPS（Cell Update Per Second）である．図 6 より，いずれの対に対しても，ρ を向上できた．1GPU 版の ρ は 44〜77 GCUPS であり，2GPU 版の ρ は 92〜128 GCUPS である．提案手法の 2GPU 版は，いずれも 1GPU 版の 1.5〜2.3 倍の速度向上率を果たしている．既存手法は 1GPU 版に対して 1.2〜1.9 倍の速度向上率に留まっているため，提案手法の枝刈りは効率のよい並列化を引き出せる．特に，速度向上率が 2 倍を超えた対 $\langle 4,5\rangle$ に対しては，超線形の結果を得た．ただし，超線形（3 倍）の速度向上はトレースバックで得られていて，提案手法が対象とする行列計算に限れば，速度向上率は提案手法で 1.0〜1.6 倍，既存手法では 1.3〜1.7 倍になっていて，両者に大きな差はない．

5.2 枝刈り率の分析

図 8 に，枝刈り率 $R = r/mn$ を示す．ここで，r は，行列計算において枝刈りした要素の数，すなわち 3.1 節の条件を満たす要素の数である．

計算スループット ρ と同様に，いずれの対に対しても提案手法は R を向上できた．1GPU 版（図 8(a)）では，対 $\langle 3,4\rangle$ に対して R が最大であり，23%の向上を果たした．同様の傾向は，他の対においても確認できる．したがって，枝刈りにより計算スループットを向上できている．

一方，対 $\langle 3,4\rangle$ に対して 2GPU 版は 49%の向上を果たしていて，1GPU 版の R よりも大きい．2GPU 版において高速化効果が大きい理由は，枝刈り対象となりうる領域を，提案手法が拡張できたためである．既存手法 [6] は，$i \geq \lceil m/2 \rceil$ もしくは $i \geq n - j$ を満たす要素 $H_{i,j}$ を枝刈り



(a) 対 $\langle 3,4\rangle$　(b) 対 $\langle 3,6\rangle$

図 7 枝刈りの対象となりうる要素の領域

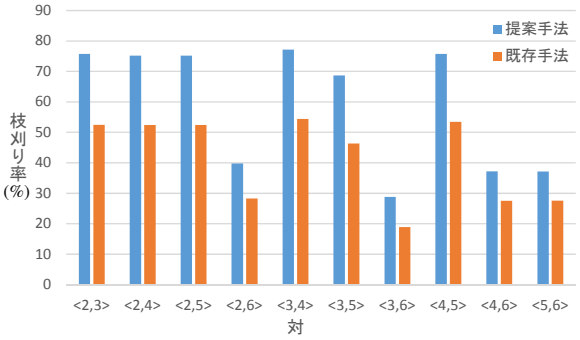
の対象とするが（3.1 節），行列 H を上下に 2 分割して各々を並列処理する SW#の 2GPU 実装は，これらの一部を枝刈りできない．具体的には，これらの部分行列におけるスコアを正しく集約するために，両者の境界に存在する行をすべて計算する必要があり，$i \geq \lceil m/2 \rceil$ を満たす要素を枝刈りできない．結果として，既存手法の 2GPU 版が枝刈り対象とできるのは，上の部分行列における $i \geq n - j$ もしくは下の部分行列における $m - i \geq j$（上の領域を上下左右を反転させたもの）を満たす要素である．一方，提案手法は，このような要素の位置関係に起因する制限を持たない．したがって，特に 2GPU 版で既存手法よりも効率のよい枝刈りを実現できた．

次に，1GPU 版における速度向上率が最大の対 $\langle 3,4\rangle$ および最小の対 $\langle 3,6\rangle$ について，アライメント結果を調べた（**表 3**）．前者で用いた下界は $L = 5,214,697$ であり，実験において最大であった．結果，枝刈りの対象にできた要素は行列全体の 77%に達し，効果的な枝刈りを実現できた（**図 7**(a)）．なお，この図では，既存手法が枝刈りしうる領域を橙色で表し，提案手法が拡張した領域を青色で表している．

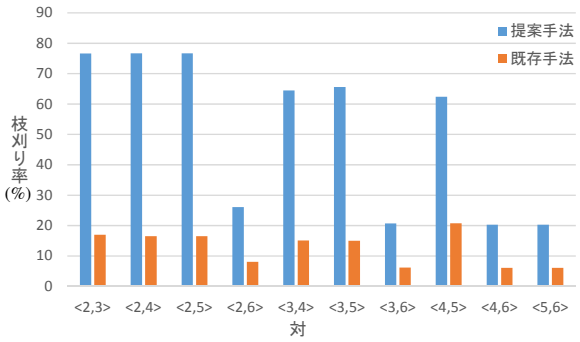
一方，後者で用いた下界は $L = 1,431,712$ であり，3 番目に小さかった．したがって，提案手法が拡張した領域は行列全体のおよそ 2%に留まり（図 7(b)），実行時間に関して大きな改善は得られなかった．なお，最小および 2 番目に小さい下界は，それぞれ対 $\langle 5,6\rangle$ および対 $\langle 4,6\rangle$ で得られていて，いずれも対 $\langle 3,6\rangle$ で得られた速度向上率（1.06 倍）と比べて大きな差はなかった．

5.3 長大な生体配列に対する適用例

より長大な生体配列に対する提案手法の有用性を調べるために，霊長類の塩基配列を用い（表 2），アライメントを実施した（表 3）．対 $\langle 7,8\rangle$ に対しては，配列 7 における類似部位は 27,961,827〜58,340,489 であった．対 $\langle 7,9\rangle$ に対しては，配列 7 における類似部位は 27,437,780〜58,535,035 であった．これらを用いて得た，ゴリラとチンパンジーのアライメントのスコアの下界は $L = -35,205,808$ であった．$L < 0$ であるため，提案手法は既存手法を高速化できなかった．



(a) 1 GPU



(b) 2 GPU

図 8 枝刈り率の比較

表 3 アライメント結果の一覧

対	スコア	不一致数	ギャップ数
⟨1, 3⟩	5,226,806	19	134
⟨1, 4⟩	5,221,028	481	1,615
⟨3, 4⟩	5,221,433	464	1,481
⟨1, 3⟩	5,226,806	19	134
⟨1, 6⟩	5,179,709	628	15,972
⟨3, 6⟩	1,440,080	231	1,990
⟨7, 8⟩	6,268,702	3,664,543	3,655,628
⟨7, 9⟩	14,383,541	1,626,256	3,780,342

6. まとめ

本論文では，塩基配列に対する全対比較の高速化を目的として，枝刈りにより SW アルゴリズムの実行時間を短縮する手法を提案した．提案手法は，計算済みの対のスコアを基に，残りの対におけるスコアの下界を導出することで，効率のよい枝刈りを実現する．

実験の結果，炭疽菌の塩基配列に対しては，提案手法は既存手法よりも 1.17 倍高速であり，2GPU 版では 1.32 倍高速であった．一方，霊長類の塩基配列に対しては，枝刈りに用いる下界が負の値であったため，高速化は達成できなかった．

今後の課題は，グローバルアライメントを求める Needleman-Wunsch アルゴリズムに対する提案手法の適用が挙げられる．系統樹の作成 [9] や，マルチプルアライメントの出力 [11] には，グローバルアライメントの出力が必要とされ，本論文で取り扱ったローカルアライメントと同様に高速化の需要が存在するためである．

謝辞 本研究の一部は，JST CREST「進化的アプローチによる超並列複合システム向け開発環境の創出」および科研費 25136711 の補助による．

参考文献

[1] Smith, T. F. and Waterman, M. S.: Identification of Common Molecular Subsequences, *J. Molecular Biology*, Vol. 147, No. 1, pp. 195–197 (1981).
[2] Li, I. T., Shum, W. and Truong, K.: 160-fold acceleration of the Smith-Waterman algorithm using a field programmable gate array (FPGA), *BMC Bioinformatics*, Vol. 8, No. 185 (2007). 7 pages.
[3] Liu, Y. and Schmidt, B.: SWAPHI: Smith-Waterman Protein Database Search on Xeon Phi Coprocessors, *Proc. 25th IEEE Int'l Conf. Application-specific Systems, Architectures and Processors (ASAP'14)*, pp. 184–185 (2014).
[4] Lindholm, E., Nickolls, J., Oberman, S. and Montrym, J.: NVIDIA Tesla: A Unified Graphics and Computing Architecture, *IEEE Micro*, Vol. 28, No. 2, pp. 39–55 (2008).
[5] de O. Sandes, E. F. and de Melo, A. C. M. A.: CUDAlign: Using GPU to Accelerate the Comparison of Megabase Genomic Sequences, *Proc. 15th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming (PPoPP'10)*, pp. 137–146 (2010).
[6] de O. Sandes, E. F. and de Melo, A. C. M. A.: Retrieving Smith-Waterman Alignments with Optimizations for Megabase Biological Sequences Using GPU, *IEEE Trans. Parallel and Distributed Systems*, Vol. 24, No. 5, pp. 1009–1021 (2013).
[7] de O. Sandes, E. F., Miranda, G., de Melo, A. C. M. A., Martorell, X. and Ayguadé, E.: CUDAlign 3.0: Parallel Biological Sequence Comparison in Large GPU Clusters, *Proc. 14th IEEE/ACM Int'l Symp. Cluster, Cloud and Grid Computing (CCGrid'14)*, pp. 160–169 (2014).
[8] Korpar, M. and Šikić, M.: SW#–GPU-enabled exact alignments on genome scale, *Bioinformatics*, Vol. 29, No. 19, pp. 2494–2495 (online), available from ⟨https://sourceforge.net/swsharp/code/⟩ (2013).
[9] Feng, D.-F. and Doolittle, R. F.: Progressive Sequence Alignment as a Prerequisite to Correct Phylogenetic Trees, *J. Molecular Evolution*, Vol. 25, No. 4, pp. 351–360 (1987).
[10] Higgins, D. G. and Sharp, P. M.: CLUSTAL: a package for performing multiple sequence alignment on a microcomputer, *Gene*, Vol. 73, No. 1, pp. 237–244 (1988).
[11] Notredame, C., Higgins, D. G. and Heringa, J.: T-Coffee: A Novel Method for Fast and Accurate Multiple Sequence Alignment, *J. Molecular Biology*, Vol. 302, No. 1, pp. 205–217 (2000).
[12] Wilbur, W. J. and Lipman, D. J.: The Context Dependent Comparison of Biological Sequences, *SIAM J. Applied Mathematics*, Vol. 44, No. 3, pp. 557–567 (1984).
[13] Gotoh, O.: An Improved Algorithm for Matching Biological Sequences, *J. Molecular Biology*, Vol. 162, No. 3, pp. 705–708 (1982).
[14] National Center for Biotechnology Information: NCBI data (online), available from ⟨http://www.ncbi.nlm.nih.gov/⟩ (2014).