GPU を用いた 3 次元 Smith-Waterman 法の高速化手法の提案

須戸 里織
Saori SUDO

1　はじめに

従来より，画像処理用のアクセラレータである GPU（Graphic Processing Unit）は，科学技術計算やシミュレーションといった汎用計算へ応用されるようになり，様々なアプリケーションが開発され，多くの研究が取り組まれている．GPU を画像処理以外の汎用計算に応用して用いることを GPGPU（General-purpose computing on GPU）[1] という．CPU に比べて多くのコア数を持つ GPU は，それらのコアを利用した並列処理で高い計算性能を発揮する．また，GPU は消費電力あたりの性能が高いという利点を持つ．GPGPU は，NVIDIA が提供する C 言語の統合開発環境である CUDA（Compute Unified Device Architecture）[2] や，並列コンピューティングのためのフレームワークである OpenCL[3] 等を利用して行うことが可能である．並列計算によって高速化が期待されている研究のひとつに，生物学の分野において DNA 配列中に同じ順序で並んでいる文字列や文字パターンを見つける配列アラインメント [4] がある．そのうち，配列アラインメントのひとつである SW（Smith-Waterman）法は，そのアルゴリズムの並列性から従来より並列計算機による高速化が試みられてきた [5] [6]．また配列アラインメントは，頭部に専用の装置を装着し近赤外光で脳血流量の変化を計測し，脳の働きを観察する光トポグラフィ [7] や，照明の照度値などの時系列データの解析といった生物学の分野以外の応用も試みられている [8]．時系列データを文字列に置き換え，SW 法を用いて類似部分を探索するという研究もみられる [9]．時系列データは，データ全体の類似部分だけでなく，複数の局所的に類似している部分を求めることが必要とされている．このため，時系列データの解析では配列アラインメントの中で局所的配列アラインメントを行う SW 法を用いている．光トポグラフィの研究では，図 1 に示すような多数の系列からなるデータの全組み合わせについて比較が必要である．そのため，SW 法で類似部分を探索するためには多くの計算を行う必要がある．また，光トポグラフィの計測点は数十チャンネルあるため，すべてのチャンネルで共通する類似部分を探索するためには，2 系列だけでなく多系列の配列アラインメントを行う必要がある．SW 法を時系列データの解析に用いるとき，多系列のデータを同時比較することと GPU による高速化が必要である．SW 法はアルゴリズムの細粒度並列性が高く，一系列あたりの文字列長が膨大でなければ GPU による高速化が可能である．本研究では，3 次元 SW 法の提案を行い，GPU で高速に計算する手法について述べる．また，実行速度を CPU と GPU で比較することで評価を行い，結果について検討した．

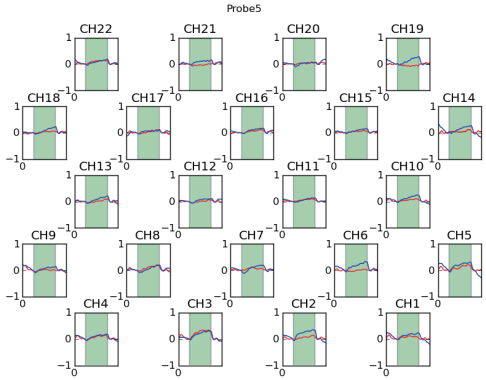

Fig.1　光トポグラフィで取得されたチャンネルごとの脳血流データ

2　配列アラインメント

2.1　概要

配列アラインメントは 2 つ以上の配列の類似部分を探索する手法である．2 つの配列を比較するペアワイズ配列アラインメント (pair-wise sequence alignment) と，3 つ以上の配列を比較する多重配列アラインメント (multiple sequence alignment) がある．また，別の分類では，配列の全体が含まれるように比較を行う大域的配列アラインメント（global sequence alignment）と配列の一部で比較を行う局所的配列アラインメント（local sequence alignment）がある．光トポグラフィや照明の照度値といった時系列データの類似部分を求める場合，データ全体の類似部分だけでなく，局所的な類似部分を求めることが必要とされている．また，多系列に共通する類似部分を探索する必要もあり，この場合に適したアルゴリズムは SW 法である．

2.2　SW（Smith-Waterman）法の概要

SW 法は 2 つの文字列の共通部分列を抽出する手法である．図 2 に示すような対象文字列 X，対象文字列 Y を，行と列に割り当てた文字列テーブルを計算することによって求められる．図 2 の左上の要素から右下の要素へとスコアを計算し，値を格納していく．文字列テーブルの右下まで計算を終えたのち，最も高いスコアの要素からスコアが 0 になるまで，要素を選択した経路を逆戻りし類似文字列を抽出する（トレースバック）．SW 法によって抽出された類似文字列の例を図 3 に示す．SW

法のスコアの計算では match，mismatch，gap というパラメータを用いる．match は 2 つの文字列が一致したとき，mismatch は不一致のとき，gap はスペースの挿入に関するスコアである．2 つの文字列の長さを m，n とすると SW 法の計算時間量と空間時間量はスコアテーブルの大きさより $O(mn)$ である．

2.3 SW（Smith-Waterman）法の計算手順

SW 法では一般的に match = 1, mismatch = -1, gap = -1 が用いられている．長さが m の配列 $X = x_1x_2...x_m$ および長さが n である配列 $Y = y_1y_2...y_n$ を対象としたときの，SW 法のアルゴリズムの流れを以下に示す．

Step1. 配列 X，Y について，文字列テーブルを作成し，それぞれの文字列を列と行に割り当てる．

Step2. i 行 j 列にある要素が $SW(i,j)$ であるとすると，$0 \leq k \leq m$ および $0 \leq l \leq n$ において，$SW(k,l) = SW(k,0) = SW(0,l) = 0$ と初期化を行う．

Step3. それぞれの要素について式（1）を基にスコアを計算する．

Step4. 最も高いスコアを持つ要素からトレースバックを行う．

$$SW(i,j) = \max \begin{cases} \begin{cases} SW(i-1,j-1) + \text{match} & \text{if}(x_i = y_j) \\ SW(i-1,j-1) + \text{mismatch} & \text{else} \end{cases} \\ SW(i-1,j) + \text{gap} \\ SW(i,j-1) + \text{gap} \\ 0 \end{cases} \quad (1)$$

文字列 “ACAC” と “AGCA” の類似部分を SW 法で求めたとき，スコアテーブルは図 4 のようになる．類似部分を抽出するために，最大のスコアの要素からスコアが 0 の要素までトレースバックを行う．

図 5 において，最大のスコアを持つ要素は $(i,j) = (4,3)$ となり，トレースバックは $(i,j) = (4,3), (i,j) = (3,2), (i,j) = (2,1)$ という経路をたどる．スコアが 0 になるとトレースバックは終了して，辿ってきた経路の逆方向に向かって戻り，スコアが 0 のところは含まずに，要素の座標に対応している文字列から類似部分を抽出する．“ACAC” と “AGCA” からは “CA” が類似度の高い部分文字列として抽出される．

3 CUDA

3.1 CUDA の概要

本研究では，NVIDIA 社が提供している GPU 向けの統合開発環境である CUDA を用いた．CUDA を用いることで GPU を並列計算機として利用することができる．CUDA では，C 言語の構文に加えて，ホストとデバイス（GPU とビデオメモリ）のメモリ確保，解放，データ転送といったデバイスの操作に関する機能の拡張が行われている．

CUDA では，ホストとデバイスの処理を分割して記述する．デバイスの処理は kernel 関数と呼ばれる関数に記述して行い，それ以外をホストで処理する．デバイスで

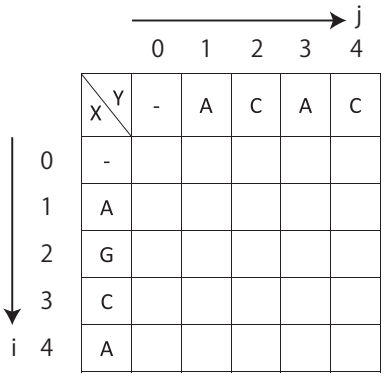

Fig.2 文字列テーブル

“PELICAN” → “ELICAN” “PAWHEAE” → “AW_HE”
“COELACANTH” → “ELACAN” “HEAGAWGHEE” → “AWGHE”

Fig.3 SW 法で抽出した類似部分列の例

X \ Y	-	A	C	A	C
-	0	0	0	0	0
A	0	1	0	1	0
G	0	0	0	0	0
C	0	0	1	0	1
A	0	1	0	2	0

Fig.4 全ての要素の計算

X \ Y	-	A	C	A	C
-	0	0	0	0	0
A	0	1	0	1	0
G	0	0	0	0	0
C	0	0	1	0	1
A	0	1	0	2	0

Fig.5 トレースバック

処理を行う際の大まかな流れを述べる．まず，ホストでデバイスのメモリ確保を行う．そして，ホストのメモリからデバイスのメモリへ計算に必要なデータを転送する．次に，デバイスの kernel 関数で転送されたデータを処理する．さらに，デバイスのメモリからホストのメモリへ計算結果のデータを転送する．最後に，ホストでデバイスのメモリ解放を行う．このように，ホストで処理を行いつつデバイスに処理を与えている．

3.2 デバイスの構造

GPU は grid，block，thread の 3 つの単位で管理される．複数の thread の集まりを block といい，複数の block の集まりを grid と定義する．1 つの grid は 1 つの GPU に対応する．GPU は複数の Streaming Multi Processor(以降 SM) からなり，1 つの SM は 1 つの block に対応する．SM は複数の Streaming Processor(以降

SP) からなり，1 つの SP は 1 つの thread に対応している．各 thread に割り当てられた処理は同時に実行され，並列処理が実現される．また，実行の際には warp という単位で同じ命令が行われ同時に処理される．1warp は 32 個の thread からなるため，thread 数は 32 の倍数の場合が望ましい．warp 内の各 thread が分岐命令で異なる方向に分岐することを繰り返していくと，並列処理が行われないため GPU の性能低下を招く．このことを warp divergent といい，プログラムを書くときは注意しなければならない．

4 関連研究

SW 法は配列アラインメントのうち精度を重視するが，速さを重視する手法には BLAST(Basic Local Alignment Search Tool)[10] や FASTA(FAST-ALL)[4] がある．一方，多重配列アラインメントを求める方法に，ペアワイズ配列アライメントで使用している DP 法を 3 つの配列に拡張する手法がある [11]．しかし，この手法では，3 配列以上では必要な計算ステップ数やメモリ量は解析する配列の数に対して，指数関数的に増加するため，文字列長が短い配列しか扱えないという問題がある．よって，一般的に多重配列アラインメントの計算には近似法が用いられている．近似法を用いた多重配列アラインメントのプログラムは CLUSTALW[12]，MAFFT[13]，MAVID[14]，T-COFFEE[15] などがある．ただし，これらは近似法を用いていない手法に比べて，配列アラインメントの精度が劣る．また，大域的配列アラインメントであるため，類似度が最大のもの以外の類似部分を抽出することができない．それに対して，SW 法は局所的配列アラインメントであるため，n 次元に拡張しても類似度が最大のもの以外の多重配列アラインメントを行うことができる．

一方，GPU で高速に配列アラインメントを行う研究 [16] [17] も多くあり，代表的なものとしては，CUDAlign[5] や CUDASW++2.0[6] などがある．SW 法の GPU 実装の多くは，SW 法のスコアテーブルを全て保持しておらず，小ブロックに分けて保持している場合が多い．これは，SW 法のスコアテーブルはデータ量が膨大になるので，デバイス側からホスト側にスコアテーブルを戻すときのデータ転送に時間がかかるためである．また，多くの実装ではデバイス側でスコアテーブルの計算をして，トレースバックはホスト側で行う場合が多い．

5 3 次元 SW 法の提案

5.1 3 次元 SW 法の概要

SW 法を用いて 3 つの配列の類似部分列を求めるために，本研究では SW 法のスコア行列を 3 次元に拡張してスコアを計算する．スコアの計算を行ったのち，2 次元のときと同様に 3 次元的にトレースバックを行う．スコア行列の計算方法およびパラメータの値は 2.3 節に記述したアルゴリズムを拡張したものを用いる．

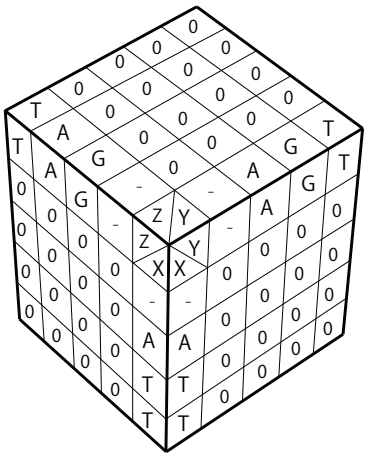

Fig.6 初期化後の 3DSW 法のスコアテーブル

5.2 3 次元 SW 法のアルゴリズム

長さが m の配列 $X = x_1x_2...x_m$ および長さが n である配列 $Y = y_1y_2...y_n$ および長さが l である配列 $Z = z_1z_2...z_l$ を対象としたときの 3 次元 SW 法のアルゴリズムの流れを以下に示す．

Step1. 配列 X，Y，Z について，文字列テーブルを作成し，それぞれの文字列を x，y，z 軸に割り当てる．

Step2. 位置 i,j,k にある要素が $SW(i,j,k)$ であるとすると，$0 \leq p \leq m$，$0 \leq q \leq n$ および $0 \leq r \leq l$ において，$SW(p,q,r) = SW(p,0,0) = SW(0,q,0) = SW(0,0,r) = 0$ と初期化を行う．

Step3. それぞれの要素について，3 つの文字を式 (2) を基にスコアを計算する．

Step4. 最も高いスコアから 3 次元的にトレースバックを行う．

$$
\begin{aligned}
&SW(i,j,k) \\
&= \max \begin{cases}
\begin{cases} SW(i-1,j-1,k-1) + \text{match} & \text{if}(x_i = y_j = z_k) \\ SW(i-1,j-1,k-1) + \text{mismatch} & \text{else} \end{cases} \\
SW(i-1,j-1,k) + \text{gap} \\
SW(i,j-1,k-1) + \text{gap} \\
SW(i-1,j,k-1) + \text{gap} \\
SW(i-1,j,k) + \text{gap} \\
SW(i,j-1,k) + \text{gap} \\
SW(i,j,k-1) + \text{gap} \\
0
\end{cases}
\end{aligned} \tag{2}
$$

図 6 では，$(i,j,k) = (3,3,3)$ において，x_3= “T”，y_3= “T”，z_3= “T” であり 3 つの文字が一致している．3 つの文字列が等しいので $SW(3,3,3)$ は式 (2) に代入すると，$SW(2,2,2) = 0$ であるので，$SW(3,3,3) = 1$ となる．

6 実装

3 次元 SW 法を GPU で高速化する際に以下の 3 種類のプログラムを用いて評価を行った．

- トレースバックを含めた GPU 版 3 次元 SW 法 (GPU-3DSWT)
- GPU でスコアテーブルを計算し CPU でトレースバックを行う 3 次元 SW 法（GPU-3DSW）

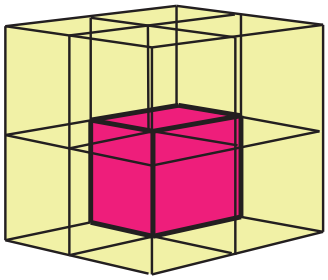

Fig.7　各要素のデータの依存性

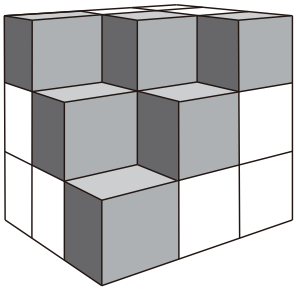

Fig.8　依存性のない要素

- トレースバックを含むシングルスレッドの CPU 版 3 次元 SW 法（CPU-3DSW）

6.1　GPU での実装の概要

3 次元 SW 法のスコアテーブルにおいて，それぞれの要素は図 7 に示すように 7 近傍の要素にデータが依存関係にある．そのため，図 8 に示すように斜めの面において，各要素は依存性を持たず，並列に計算が可能である．また，図 9 に各ステップごとに計算可能な要素を示す．このような SW 法の並列性を CUDA を利用して 3 次元 SW 法の GPU 実装を行う．

6.2　GPU での実装の手順

図 9 に示す要素を並列に計算するために，CUDA における thread を図 10 のように割り当てる．一つの thread でスコアテーブルの一行を計算する．図 10 では，step=3 のときに最大 7 つのスレッドが並列に実行されている．また，block の分割においては，一つの thread で，スコアテーブルの一行を計算するため，平面上に block を割り当てることと同等になる．図 11 に示すように，block の x 座標と y 座標を足したものが現在の step 数と等しいときに，そのブロックが実行されるようになっている．なお，今回の実装ではシェアード・メモリは使用していない．

6.3　トレースバックの実装

GPU が分岐命令に時間がかかる性質があるので，通常 SW 法の GPU 実装では，トレースバックは CPU 側で行われている．しかし，3 次元にスコアテーブルを拡張すると，データ量が膨大になるので，スコアテーブルをデバイス側からホスト側へ戻してくるときに時間がかかる．よって，本研究ではデバイス側でトレースバックを行う．

デバイス側でトレースバックを行うために，スコアテーブルはスコアと参照要素の方向を保持しておく．全てのスコアテーブルの計算を終えたのち，デバイス側でスコアテーブルに保持していたスコアと参照要素をもとにトレースバックを行い，抽出された共通部分列のみをホスト側に戻すよう実装した．

7　評価

3 次元 SW 法を表 1 のような構成のマシンを用いて評価する．

SW 法のスコアテーブルを 3 次元に拡張しているため，

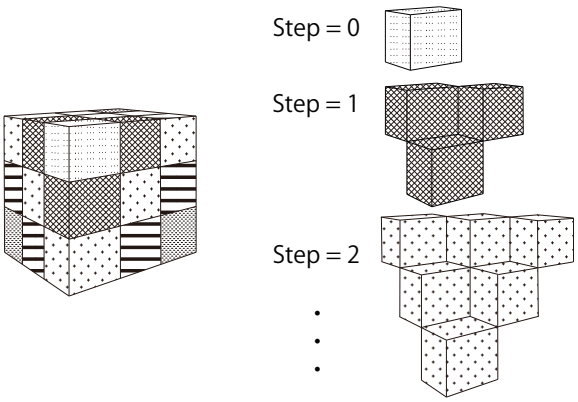

Fig.9　各ステップごとに独立であり並列計算可能な要素

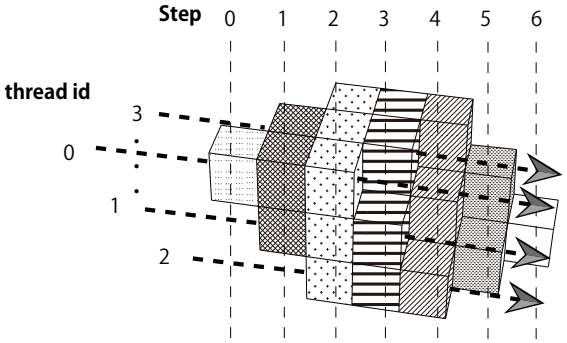

Fig.10　GPU-3DSW の thread の計算手順

Table1　使用マシン環境

	マシン 1	マシン 2
CPU	Intel Xeon W3530 2.80GHz	Intel Core i5-2400 3.10GHz
GPU	Tesla C2050	GeForce GTX 460
Memory	6GB	8GB
OS	Debian 5.0.6	Ubuntu 11.04
開発環境	CUDA 3.1	CUDA 3.2
コンパイルオプション	-O3	-O3

データ量が増加した場合，GPU のグローバルメモリの容量から，DNA 配列のような文字列長の長いものは扱えない．3 次元 SW 法は，光トポグラフィの信号値や照明の照度値解析を主な利用対象とする．これらの利用では，ある一定の時間の間に局所的な類似部分を探索できればよいため，ここでは長い文字列長の探索は必要ではない．図 12 にマシン 1 にについて，図 13 にマシン 2 について CPU-3DSW，GPU-3DSW，GPU-3DSWT の

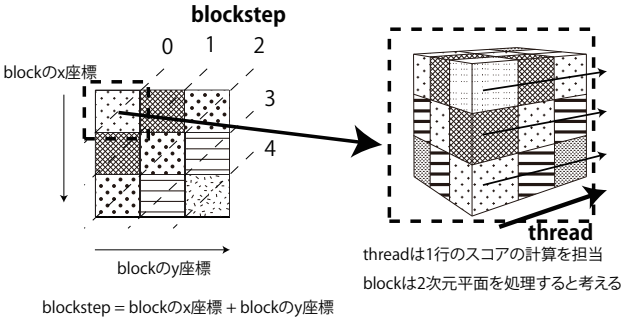

Fig.11　GPU-3DSW の block の計算手順

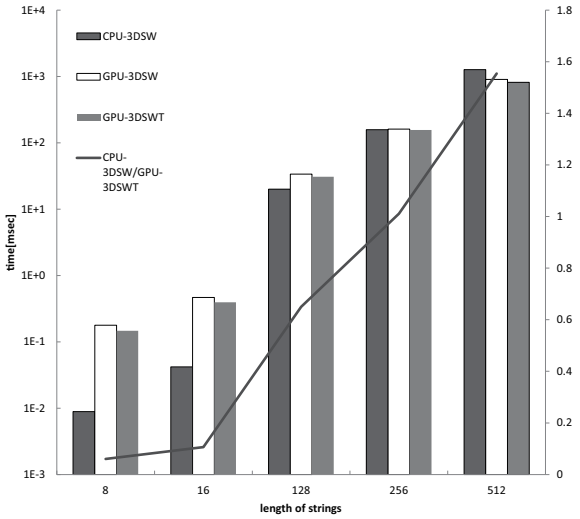

Fig.12　マシン 1 において手法ごとの実行時間

評価結果を示す．CPU-3DSW と GPU-3DSWT の実行時間を比較すると，文字列長が 512 の場合，マシン 1 では CPU の最大 1.5 倍，マシン 2 では CPU の最大 0.8 倍となった．また，文字列長が短い場合は GPU-3DSW と GPU-3DSWT のどちらも実行時間が遅くなっている．これは，デバイスとホスト間のデータの転送を行うのにかかるオーバーヘッドのためである．また，GPU-3DSWT では GPU 側でのトレースバックにおいて条件演算子を多用しているために，warp divergent が発生していることも考えられる．

8　まとめと今後の課題

本研究報告では 3 次元 SW 法の提案を行った．CUDA で 3 次元 SW 法を実装し GPU で実行したところ，CPU 版の 3 次元 SW 法に比べて最大 1.5 倍の高速化を確認した．

今後に考えられる課題としては，kernel 関数で条件演算子を減らし効率的にコアレッシングが起こるようにすることで，kernel 関数の実行時間を削減することが考えられる．また，3 次元だけでなく n 次元に SW 法を拡張する必要もある．n 次元 SW 法はデータ量がさらに増えるが，SW 法のスコアテーブルは類似度が低いもの同士の比較であれば，多くのスコアが 0 となるので，疎行列

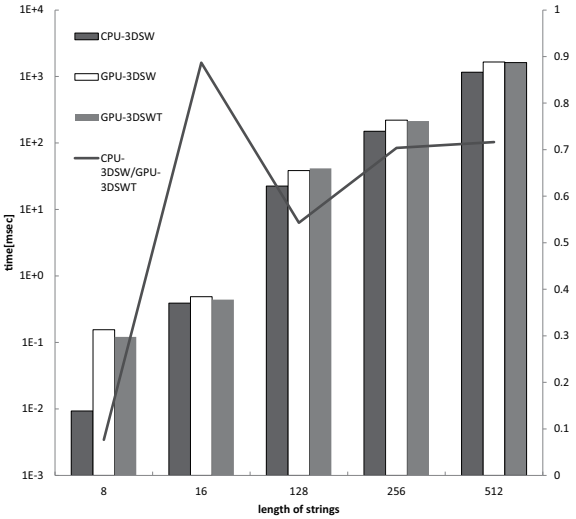

Fig.13　マシン 2 において手法ごとの実行時間

を用いてデータ量の削減を行うことも考えられる．スコアテーブルの計算の手順も，今の実装では 1 つの要素ごとに計算しており効率が悪いので，シェアード・メモリで一度にまとめて計算できるような実装を考えるべきである．また，文字列長が長いもののアラインメントを行えるようにする必要がある．

参考文献

1) John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Kruger, Aaron Lefohn, and Timothy J. Purcell. A Survey of General-Purpose Computation on Graphics Hardware. In *Eurographics 2005, State of the Art Reports*, pp. 21–51, August 2005.
2) NVIDIA. *Compute Unified Device Architecture Programming Guide*. 2007.
3) John E Stone, David Gohara, and Guochun Shi. OpenCL: A parallel Programming Standard for Heterogenous Computing Systems. *Computing in Science Engineering*.
4) David W Mount. バイオインフォマティクス ゲノム配列から機能解析へ 第 2 版. メディカル・サイエンス・インターナショナル, 2005.
5) Edans Flavius de O Sandes and Alba Cristina M A de Melo. Smith-Waterman Alignment of Huge Sequences with GPU in Linear Space. *2011 IEEE International Parallel & Distributed Processing Symposium*, Vol. 25, pp. 1199–1211, May 2011.
6) Yongchao Liu, Bertil Schmidt, and Douglas L Maskell. CUDASW++2.0: enhanced Smith-Waterman protein database search on CUDA-enabled GPUs based on SIMT and virtualized SIMD abstractions. *BMC Research Notes*, Vol. 3, No. 1, pp. 1–12, April 2010.

7) 松下晋, 中川匡弘. 光トポグラフィーによる感性情報解析 (ヒューマンコミュニケーション). 電子情報通信学会論文誌. A, 基礎・境界, Vol. 88, No. 8, pp. 994–1001, 2005-08-01.
8) 廣安知之, 西井琢真, 吉見真聡, 三木光範, 横内久猛. 相同性検索を用いた 2 つの時系列データからの類似部分抽出手法と DTW による類似部分の評価. 情報処理学会研究報告. MPS, 数理モデル化と問題解決研究報告, Vol. 2010, No. 24, pp. 1–6, 2010-09-21.
9) Tomoyuki Hiroyasu, Takuma Nishii, Masato Yoshimi, Mitsunori Miki, and Hisatake Yokouchi. The proposal of optical topograhy analizing sysytem and evaluation. 同志社大学理工学研究報告, August 2011.
10) NCBI. *BLAST Basic Local Alignment Search Tool*. 2009.
11) Yongchao Liu, Bertil Schmidt, and Douglas L. Maskell. MSA-CUDA: Multiple Sequence Alignment on Graphics Processing Units with CUDA. *Application-Specific Systems, Architectures and Processors, IEEE International Conference on*, Vol. 0, pp. 121–128, 2009.
12) Julie D Thompson, Desmond G Higgins, and Toby J Gibson. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, Vol. 22, No. 22, pp. 4673–4680, April 1994.
13) Kazutaka Katoh, Kazuharu Misawa, Keiichi Kuma, and Takashi Miyata. MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform. *Nucleic Acids Research*, Vol. 30, No. 14, pp. 3059–3066, July 2002.
14) Nicolas Bray and Lior Pachter. MAVID: constrained ancestral alignment of multiple sequences. *Genome Research*, Vol. 14, No. 4, pp. 693–699, April 2004.
15) C Notredame, Desmond G. Higgins, and Jaap Heringa. T-Coffee: A Novel Method for Fast and Accurate Multiple Sequence Alignment. *Journal of Molecular Biology*, Vol. 14, No. 4, pp. 693–699, April 1994.
16) 土肥慶亮, 柿本雄, 柴田裕一郎, 濱田剛, 小栗清. GPU クラスタにおける Smith-Waterman アルゴリズムの実装手法の提案. *SACSIS2010*, Vol. 2010, No. 5, pp. 209–216, April 2010.
17) 宗川裕馬, 伊野文彦, 荻原兼一. 統合開発環境 CUDA を用いた GPU での配列アライメントの高速化手法. 情報処理学会研究報告, Vol. 114, No. 19, pp. 13–18, March 2008.