Bulletin of Networking, Computing, Systems, and Software – www.bncss.org, ISSN 2186–5140
Volume 3, Number 1, pages 39–43, January 2015

Accelaration of the Smith-Waterman Algorithm with traceback on the FPGA

Kaoru Hashimoto, Koji Nakano, and Yasuaki Ito
Department of Information Engineering
Hiroshima University
Kagamiyama 1-4-1, Higashi Hiroshima, 739-8527 Japan

***Abstract*—The main contribution of this paper is to propose an FPGA implementation of Smith-Waterman algorithm with the traceback. A homology search for genes in the bioinformatics is a fundamental problem to understand these functions. Smith Waterman algorithm is a technique for finding local alignments of a DNA sequence (string) which match a pattern string. We have implemented the Smith-Waterman module with 1024 pattern length in a Xilinx Virtex-7 FPGA XC7VX485T-2FFG1761. The implementation runs in approximately 154MHz clock frequency. In the case of the pattern length is 1024 and the input length is 256k, the computing time of our FPGA implementation is 1822 times faster than the CPU implementation.**

***Index Terms*—Bioinformatics, Smith-Waterman Algorithm, Block RAM, FPGA**

I. はじめに

本節では Smith-Waterman アルゴリズム [1] で検出されるローカルアラインメントついて説明する．後半では FPGA について説明する．

バイオインフォマティクスの分野では，DNA 塩基配列やタンパク質のアミノ酸配列の機能を既知の配列から特定または類推するための方法が考案されている．DNA 配列の機能を調べる方法として，既知の配列と類似度を比較する方法がある．類似度が高ければ，調査対象である配列が既知の配列と同様な機能を持っている可能性が高い．これにより的を絞った検査や実験を行うことができる．以降，配列は文字列と同義であるとし，配列の要素を文字と表現する．

配列間の類似度を求める方法とその結果を配列アラインメントと呼ぶ．配列アラインメントとは，配列間の要素の対応関係を計算すること，またその計算結果を指している．2 つの DNA 塩基配列を例に挙げると，それぞれの配列の中にギャップ記号 – を挿入し，長さを揃えたものが配列アラインメントである．ギャップの挿入は，DNA 配列への挿入や欠損など進化の過程における DNA の変化と対応している．具体的な配列アラインメントの結果を図 1 に示す．図 1 では，長さの異なる 2 つの入力配列をアラインメントした結果が示されている．また，類似度は関数 $w(x, y)$ を用いて定義する．

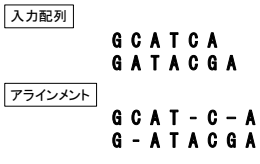

Fig. 1. アラインメントの例

関数 w は，文字 x, y を入力として，任意の実数値を出力する関数である．文字 x, y が $x \neq y$ であるとすると，関数 w は，$w(x, x) = match$, $w(x, y) = mismatch$, $w(x, y) = w(y, x)$, $w(x, -) = w(-, y) = -gap$ を満たす．$match$, $mismatch$, gap は正の整数である．gap はギャップ文字を挿入する場合のペナルティと見なすことができる．長さ l のギャップ文字を含むアラインメント $A = a_1a_2...a_l, B = b_1b_2...b_l$ を入力として，$w(x, y)$ を用いたとき類似度は式 1 によって定義される．

$$\sum_{i=1}^{l} w(a_i, b_i) \tag{1}$$

類似度が最大となる配列アラインメントは，最適アラインメントと呼ばれている．一般に配列アラインメントは最適アラインメントを求める．

また，配列アラインメントには大域アラインメントとローカルアラインメントの 2 つがある．大域アラインメントとは，2 つの配列全体の最も類似度の高いアラインメントを検出する方法とその結果である．この方法では，文字数がほぼ同じで類似した配列間を比較するのに有用である．一方で，ローカルアラインメントは 2 つの配列間で類似度の高い一部分のアラインメントを検出する方法とその結果である．長い配列同士の比較は，全体としては必ずしも類似していない場合が多いが，ある連続した部分のみ類似しているという場合は少なくない．つまり長い配列と短い配列を比較するような配列アラインメントを求める場合は，部分的類似を求めることができるローカルアラインメントの計算を行う．ローカルアラインメントの例を図 2 に示す．すなわち，求める問題は入力配列とパターン配列の類似度 (式 1) を最大化するローカルアラインメントを求めることである．

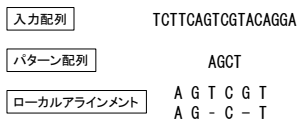

Fig. 2. ローカルアラインメントの例

FPGA は任意の回路に書き換え可能な集積回路である．回路設計者はハードウェア記述言語などを用いて回路を設計し，論理合成と配線配置ツールを経て回路データが生成される．この回路データを FPGA にダウンロードさせ動作させる．バグが存在すればチップを交換することなく修正する

ことができる．ここで，Xilinx FPGA の一般的な構成を図 3 に示す．一般的な FPGA は CLB(Configurable Logic Block)，メモリ専用のハードマクロである Block RAM，乗算や積和回路を搭載したハードマクロである DSP Slice，入出力回路やバッファ，内部配線などで構成されている．基本的には CLB に含まれるルックアップテーブルとフリップフロップを用いて組み合わせ回路と順序回路を実現している．

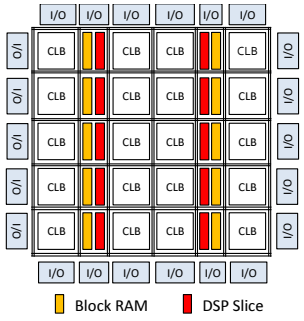

Fig. 3. FPGA の基本構造

II. SMITH-WATERMAN アルゴリズム

本節では，Smith-Waterman アルゴリズム [1] について説明する．類似度の計算方法である式 2 とトレースバックについて説明する．

Smith-Waterman アルゴリズムは，動的計画法を使用して 2 つの文字列間でのローカルアラインメントを検出するアルゴリズムである．式 2 に Smith-Waterman アルゴリズムの式を示す．長さ m のパターン文字列 $A = a_1a_2a_3...a_m$ と，長さ n の入力文字列 $B = b_1b_2b_3...b_n$ が入力として与えられている．式 2 によって求められた類似度が $(m,\ n)$ 行列 H に格納される．また，パラメータ gap, $match$, $mismatch$ はそれぞれ任意の正の整数である．

$$
\begin{aligned}
H(i,0) &= 0 \qquad \forall i \in [1,...,m] \\
H(0,j) &= 0 \qquad \forall j \in [1,...,n] \\
H(i,j) &= \max \begin{cases} 0 \\ H(i-1,j-1) + w(a_i,b_j) \\ H(i-1,j) - gap \\ H(i,j-1) - gap \end{cases} \qquad (2)
\end{aligned}
$$

$$
w(x,y) = \begin{cases} match & \text{if } x = y \\ -mismatch & \text{if } x \neq y \end{cases} \qquad (3)
$$

式 2 の意味について説明する．ここで，図 1 の 2 つのアラインメントで，上のものを入力文字列側のアラインメント，下のものをパターン文字列のアラインメントと呼ぶことにする．このアルゴリズムでは，比較する文字 a_j と b_j に注目して $H(i,j)$ の類似度を求めている．$H(i-1,j-1)+w(a_i,b_j)$ は，パターン文字列 $[a_0\ ...\ a_{i-1}]$ と入力文字列 $[b_0\ ...\ b_{j-1}]$ まで計算して求めた類似度に，パターン文字列のアラインメントに a_i を入力文字列のアラインメントに b_j の文字を追加し比較した結果を足した類似度である．次に，$H(i-1,j)-gap$ はパターン文字列 $[a_0\ ...\ a_{i-1}]$ と入力文字列 $[b_0\ ...\ b_j]$ まで計算して求めた類似度に，パターン文字列のアラインメントに a_i を入力文字列のアラインメントにギャップ文字を追加する場合の類似度である．最後に $H(i,j-1)-gap$ はパターン文字列 $[a_0\ ...\ a_i]$ と入力文字列 $[b_0\ ...\ b_{j-1}]$ まで計算して求めた類似度に，パターン文字列のアラインメントにギャップ文字を入力文字列のアラインメントに b_j を追加する場合の類似度である．負の値にならないように 0 も含めて，最大値を選択する．逐次処理における Smith-Waterman アルゴリズムの計算量は $O(mn)$ である．

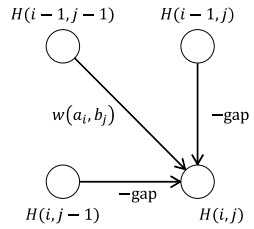

Fig. 4. 再帰式の図

また，トレースバック用の方向を保存する $(m,\ n)$ 行列 D に，$H(i-1,j-1)$, $H(i,j-1)$, $H(i-1,j)$ の中で最大値の元になった方向が次の条件に基づいて保存される．最大値が等しくなる場合は，優先順の高い方向が選択される．優先順は任意に設定することができる．以降，行列 D を方向テーブルとも呼ぶ．

if $H(i,j) = 0$ **then**
  $D(i,j) \leftarrow$ ' $\emptyset$ '
else if $H(i,j) = H(i-1,j-1) + w(a_i,b_j)$ **then**
  $D(i,j) \leftarrow$ ' $\searrow$ '
else if $H(i,j) = H(i-1,j) - gap$ **then**
  $D(i,j) \leftarrow$ ' $\downarrow$ '
else if $H(i,j) = H(i,j-1) - gap$ **then**
  $D(i,j) \leftarrow$ ' $\rightarrow$ '
end if

図 5 にローカルアラインメントとトレースバックの例を示す．

トレースバックとは，ローカルアラインメントを求める逐次処理である．本稿におけるトレースバックの実現方法は，式 2 により選択された方向をたどる方法とする．トレースバックの処理は，H 行列中の類似度の最大値 $\max\{H(i,j)\}$ の位置 (i_{max}, j_{max}) から開始される．位置 (i_{max}, j_{max}) から類似度の元となった方向をたどり，方向が $\emptyset$ になるまで続けられる．トレースバック処理をアルゴリズム 1 に示す．D 行列, i_{max}, j_{max}, 配列 $A = a_1a_2...a_m$, $B = b_1b_2...b_n$ を入力とし，ローカルアラインメント S_A, S_B を出力する．S_A, S_B は，それぞれパターン文字列のアラインメントと入力文字列のアラインメントを示す．また，S^{-1} は配列の反転を表している．例えば，$S = \text{ATG} - \text{GC}$ であるとき，$S^{-1} = \text{CG} - \text{GTA}$ である．図 5 にトレースバックによって求めたローカルアラインメントの例を示す．行列中の最大値である 5 からトレースバックが開始される．この結果ローカルアラインメント S_A, S_B が求まる．2 つの配列間で最も類似している部分が S_A, S_B によって示されている．

本研究ではパターン文字や入力文字は DNA 塩基対 (A, T,

Algorithm 1 Traceback

$k \leftarrow 0,\ i \leftarrow i_{max},\ j \leftarrow j_{max}$
while $D(i,j) \neq \emptyset$ **and** $i \neq 0$ **and** $j \neq 0$ **do**
  if $D(i,j) =$ ' $\searrow$ ' **then**
    $S_A[k] \leftarrow a_i,\ S_B[k] \leftarrow b_j,\ i \leftarrow i-1,\ j \leftarrow j-1$
  else if $D(i,j) =$ ' $\downarrow$ ' **then**
    $S_A[k] \leftarrow a_i,\ S_B[k] \leftarrow$ '-', $i \leftarrow i-1$
  else if $D(i,j) =$ ' $\rightarrow$ ' **then**
    $S_A[k] \leftarrow$ '-', $S_B[k] \leftarrow b_j,\ j \leftarrow j-1$
  end if
  $k \leftarrow k+1$
end while
$S_A \leftarrow S_A^{-1},\ S_B \leftarrow S_B^{-1}$

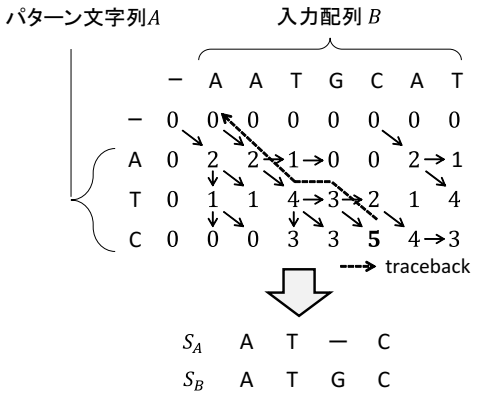

Fig. 5. ローカルアラインメントとトレースバックの例

G, C) に限定している．また，Smith-Waterman アルゴリズムは BLAST[2] のようなヒューリスティックなアルゴリズムに比べて計算量が大きいことから，FPGA[3], [4], [5] や GPU[6] など並列性の高いシステムで高速計算をさせる研究が行われている．

III. 類似度の並列計算

本節では Smith-Waterman アルゴリズムの類似度を計算する方法について説明する．長さが m の検出したいパターン文字列 $X = x_1x_2...x_m$ で入力文字列 $Y = y_1y_2...y_n$ 間の最適となるローカルアラインメントを求めたい．m 行 n 列となる類似度行列 H を求める回路を単純に実装する場合について考えてみる．その場合，式 2 で表される Smith-Waterman アルゴリズムを計算する回路を mn 個並べなければならない．さらに，この実装方法は回路規模 $O(mn)$ となる．各回路は自身の計算が終わった時点でそれ以上動作する必要はなく，FPGA のハードウェアリソースをすぐに消費するため非効率である．

Smith-Waterman アルゴリズムを逐次的に解く方法を説明する．図 4 より，3 つの値 $H(i-1,j-1), H(i-1,j), H(i,j-1)$ がすでに計算済みになっている必要がある．計算の一番初めにこの 3 つの値が揃っているのは $H(1,1)$ であり，ここから計算が開始される．行方向に計算を進める場合，$H(1,2)$ から $H(1,n)$ まで計算を進める．次に $H(2,1)$ から $H(2,n)$ まで計算を進めるといったように 1 行ずつ計算することができる．逐次処理での計算量は $O(mn)$ である．

次に，並列計算は図 6 のように斜め方向の類似度の計算を並列に行う．Smith-Waterman アルゴリズムは，STEP$(i-1)$

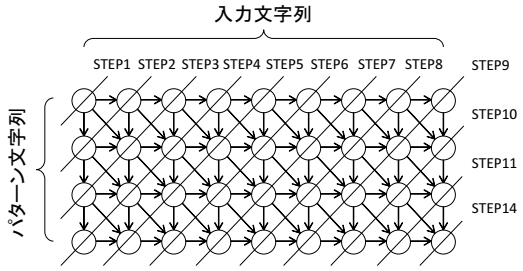

Fig. 6. Smith-Waterman アルゴリズムの並列計算

までのすべての類似度が計算済みならば STEPi の類似度を並列に求めることができる．すなわち，並列計算の計算量は $O(m+n)$ である．STEPi について求める場合，STEPi の斜線上の類似度はそれぞれ図 4 のような 3 つの値を必要とする．すなわち，STEP$(i-1)$ と STEP$(i-2)$ の 2 列を記憶するだけで充分である．

ハードウェアで実現する方法について説明する．図 6 の STEP はクロックサイクルと対応している．式 2 の類似度を計算する回路をパターン文字数である m 個用意し，それぞれの回路について 2 クロックサイクル前までの計算結果をレジスタに保持している．それぞれのレジスタは，1 ステップ前に求めた類似度と 2 ステップ前に求めたレジスタと対応している．このように実装した回路を図 7 に示す．図 7 において，各類似度計算回路には対応するパターン文字が入力されている．回路はパイプライン化され，1 クロックごとに入力文字列 Y から 1 文字ずつ入力され類似度を出力する．また，類似度の元になった方向を出力する．回路が出力する方向は後述するトレースバックのために方向テーブルに保存される．なお，行列中の最大値は次のように求める．初期値が 0 の最大値レジスタを用意し暫定最大値としておく．毎クロックサイクル出力される類似度の中の最大値と比較し，暫定最大値より大きければ，最大値レジスタを更新する．また最大値レジスタは，最大値の位置 (i_{max}, j_{max}) も記憶する．最大値が更新される度に位置も書き換えられる．

類似度計算回路のビット幅は，類似度パラメータ $match$ とパターン文字数 m で決定される．類似度の最大値は $m \times match$ であるから，最大値のビット幅は $\lfloor \log_2(m \times match + 1) \rfloor$ だけあればよい．$match = 5$ のとき $m = 512$ であっても計算に必要なビット幅は 12 ビットあれば十分である．

IV. 方向テーブルの実装

本節では FPGA 上でトレースバックに必要な方向テーブルの実装方法について説明する．はじめに，すべての計算結果を保存する場合について述べる．

パターン文字数 m，入力文字数 n とするとすべての類似度は mn 個である．これらすべてを保存するような実装方法であるが，入力文字数が FPGA の組み込みメモリによって制限されてしまう．例えば，1 つの類似度が 1byte で表現する場合であったも，パターン文字数が 1024 で入力文字数

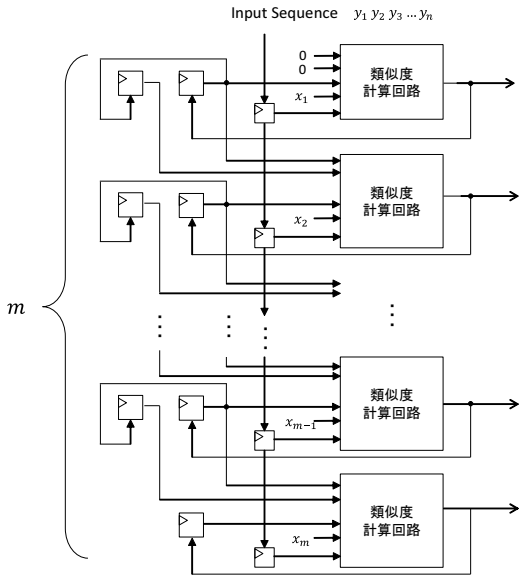

Fig. 7. 類似度計算回路

が 4M であれば 4Gbyte の容量を要する．また，2bit の方向を保存する方法であったとしても，別途 1Gbyte の容量が必要である．いずれにしても大容量なメモリが必要である．

入力文字数を制限しない方法について記述する．方向テーブルのサイズはパターン文字数 m と類似度計算のパラメータから決定することができる．トレースバックが最大となる場合の長さは式 4 で表される．これは，入力文字とパターン文字が一致して $match$ 分加算され 1 つ右下に移動する．そして，類似度が 0 ならないように右方向に進む距離は $\frac{match}{gap}$ である．そして，新たに一致する文字が出現し 1 回だけ左下方向に移動する．このことが $(m-1)$ 回行われる場合である．つまり，式 4 がトレースバックの長さの上限である．ただし，この場合の最大値となる類似度の最大値は非常に小さくなるので，このようなトレースバックが起こる可能性は小さい．

$$(1+\left\lceil\frac{match}{gap}\right\rceil)(m-1) \tag{4}$$

次に，方向テーブルの書き込みと読み出しについて説明する．方向テーブルには m 個の類似度計算回路が出力する 2bit の方向を入力とし，書き込みアドレスの番地に書き込む．一方でトレースバックは，最大値が検出されたとき最大値の位置 (i_{max}, j_{max}) から開始される．(i_{max}, j_{max}) は方向テーブルの位置と対応している．つまり，計算が進行し方向テーブルの更新が行われながら，トレースバックが行われる．これを実現するためには，方向テーブルの更新がトレースバックにて読み出す方向を上書きしないことと，書き込みと読み出しのアドレスが衝突しないようにする必要がある．このため，方向テーブルは式 4 のちょうど 2 倍の長さが必要となる．よって，方向の保存と同時にトレースバックを行うことができる．すなわち，式 4 の 2 倍のメモリアドレスの長さがあれば，トレースバックが最長となる場合であっても，読み出しは方向を保存するための書き込みと衝突することはなく，ローカルアラインメントを求めるのに十分な方向のみ保存しておくことができる．つまり，最大値が発見された場合，トレースバックに用いられるテーブルの領域は保持しつつも，トレースバックに用いられない領域を更新することが可能である．このようにして，入力文字数を制限しないトレースバックが可能である．方向テーブルの例を図 8 に示す．この例では入力文字数がいくら長いものであっても，トレースバックに必要なメモリはわずか 18 ワード分あれば十分であることを示している．

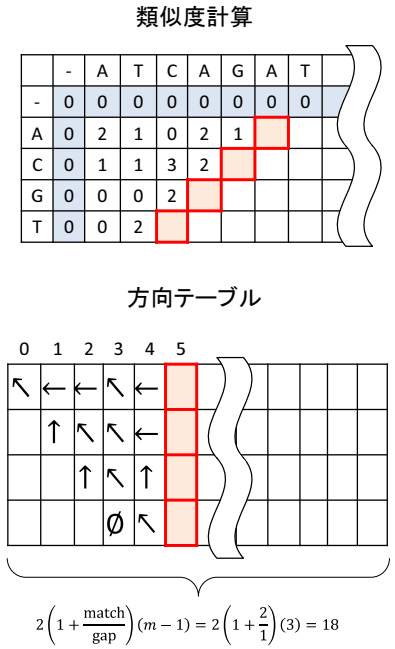

Fig. 8. 方向テーブルの例

方向テーブルはデュアルポートを備えた Block RAM を用いて実装される．方向テーブルを図 9 に示す．回路の動作

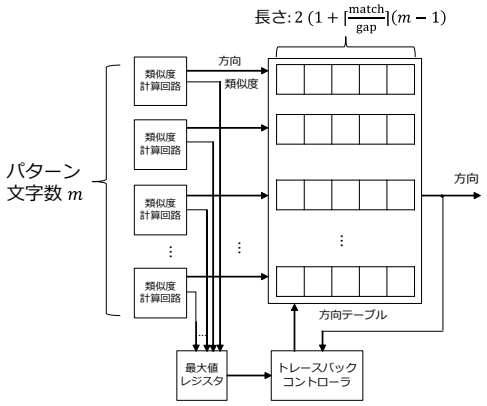

Fig. 9. 方向テーブルの回路

について説明を行う．方向テーブルのアドレス入力はそれぞれ方向の書き込み専用アドレス，トレースバックによる読み出し専用アドレスに対応している．類似度計算回路から出力される方向は，常に書き込みアドレスへ書き込みを

行う．一方でトレースバックは，最大値レジスタが更新されたときに開始される．図 9 のトレースバックコントローラは，アルゴリズム 1 を実行する．最大値レジスタが更新されると，最大値の位置 (i_{max}, j_{max}) に対応した読み出しアドレスが方向テーブルに入力され，方向テーブルが出力した方向をトレースバックコントローラに戻し，読み出しアドレスを更新していく．

V. 回路全体の動作

入力配列が 1 文字ごとに図 7 に入力される．類似度計算回路が類似度と方向を出力する．同時に，類似度計算回路から出力されるの方向を方向テーブルに保存する．また，最大値レジスタがもつ暫定最大値より大きな類似度が出力された場合，最大値レジスタを更新する．最大値レジスタが更新されるたびに，最大値の位置と対応するトレースバックテーブルから方向を読み出してトレースバックを実行する．最後に更新された最大値の位置とそのトレースバックによって出力された方向の列，パターン文字列と入力文字列によって，ローカルアラインメントが求められる．

VI. 実装結果

Xilinx Virtex-7 FPGA(XC7VX485T-2FFG1761) に Smith-Waterman アルゴリズムを実装した．実装環境は Xilinx ISE 14.7 を使用し，Verilog HDL でハードウェア設計を行った．最大動作周波数を表 I，FPGA のフリップフロップ (FF) とルックアップ (LUT) の使用率を表 II に示す．また，$match = 2$, $gap = 1$ の場合の，それぞれのパターン文字数のメモリアドレスの長さと Block RAM 使用率を表 III に示す．なお，使用した FPGA には 2060 個の 18Kbit Block RAM が搭載されている．

TABLE I
最大動作周波数

パターン文字数	動作周波数
32	212.48
64	199.13
128	199.33
256	190.70
512	190.00
1024	154.62

TABLE II
リソース使用率

パターン文字数	FF	LUT
32	0.57%	1.41%
64	1.33%	3.04 %
128	1.88%	5.93 %
256	4.21%	13.70 %
512	9.35%	36.08 %
1024	19.92%	84.96 %

VII. 性能比較

パターン文字数 1024 で入力文字数 262144 の場合の CPU と FPGA の計算時間はそれぞれ，3.150[s]，1.729[ms] となった．ここで CPU は Intel Corei5-3470 で，FPGA は Xilinx Virtex7 FPGA XC7VX485T-2FFG1761 を用いた．CPU と比較し FPGA ではおよそ 1822 倍の高速化が実現可能である．

TABLE III
メモリアドレスの長さと BLOCK RAM 使用率

パターン文字数	テーブル長	使用数	使用率
32	186	4	0.19 %
64	378	8	0.39 %
128	762	15	0.73 %
256	1530	58	2.81 %
512	3066	171	8.30 %
1024	6138	684	33.2 %

VIII. まとめ

本研究では，SmithWaterman アルゴリズムを FPGA に実装した．パターン文字数にのみ依存した方向テーブルを構成することで，入力文字数に制限なくトレースバックの実行が FPGA 上で可能になった．今後の課題としては，文献の実装方法との詳細な比較，アミノ酸配列への対応，FPGA の内部構造を意識した回路の最適化 (動作周波数の改善等)，さらに実システムの構築などが挙げられる．

REFERENCES

[1] T. Smith and M. Waterman, “Identification of common molecular subsequences,” *Journal of Molecular Biology*, vol. 147, no. 1, pp. 195 – 197, 1981.

[2] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman, “Basic local alignment search tool,” *Journal of Molecular Biology*, vol. 215, pp. 403–410, 1990.

[3] L. Hasan, Z. Al-Ars, Z. Nawaz, and K. Bertels, “Hardware implementation of the Smith-Waterman algorithm using recursive variable expansion,” *Design and Test Workshop. IDT 2008. 3rd International*.

[4] Z. Nawaz, M. Nadeem, H. van Someren, and K. Bertels, “A parallel FPGA design of the Smith-Waterman traceback,” in *Proceedings of the International Conference on Field-Programmable Technology, FPT 2010*. IEEE, 2010, pp. 454–459.

[5] Y. Yamaguchi, H. K. Tsoi, and W. Luk, “FPGA-based Smith-waterman algorithm: Analysis and novel design,” in *Proceedings of the 7th International Conference on Reconfigurable Computing: Architectures, Tools and Applications*, ser. ARC’11, 2011, pp. 181–192. [Online]. Available: http://dl.acm.org/citation.cfm?id=1987535.1987561

[6] A. Khajeh-Saeed, S. Poole, and J. Blair Perot, “Acceleration of the Smith-Waterman algorithm using single and Multiple Graphics Processors,” *J. Comput. Phys.*, vol. 229, no. 11, pp. 4247–4258, Jun 2010. [Online]. Available: http://dx.doi.org/10.1016/j.jcp.2010.02.009