情報処理学会論文誌 Vol.55 No.3 1167–1176 (Mar. 2014)

推薦論文

FPGAを用いたBLASTアルゴリズムの高速化

石川 淑[1] 田中 飛鳥[1] 宮崎 敏明[1,a)]

受付日 2013年6月27日, 採録日 2013年12月4日

概要：Basic Local Alignment Search Tool（BLAST）は最も有名なシーケンスアライメントツールの1つである．シーケンスアライメントとはタンパク質（またはDNA）データベースから検索対象となるタンパク質（またはDNA）配列を列挙することであり，配列どうしの類似部分検索のために使用される．シーケンスアライメントは，生物学上の進化や遺伝子系図を調べるうえで重要であることから，バイオインフォマティクス分野では欠かせない情報である．そのため，従来からハードウェアを用いて，BLASTを高速化する試みがなされてきた．BLASTは前処理，seeding，ungapped extension，gapped extension，traceback処理から構成される．従来のハードウェア化は，gapped extension処理が中心であり，他の部分は，ホストマシン上で処理される形態であった．本論文では，前処理，traceback処理を含むすべてのBLAST処理をハードウェア化し，ホストマシン上のソフトウェア処理との処理速度のアンバランスを解消することによりBLAST全体の高速化を行う．提案回路を廉価なFPGA（Field Programmable Gate Array）に実装した結果，ソフトウェア実装に比べ約790倍の高速化を達成した．また，gapped extension処理とtraceback処理に対してさらなる高速化手法を提案し，840倍以上の高速化を実現した．

キーワード：FPGA，BLASTP，Smith-Waterman，シーケンスアライメント

Accelerating BLAST Algorithm Using an FPGA

Shizuka Ishikawa[1] Asuka Tanaka[1] Toshiaki Miyazaki[1,a)]

Received: June 27, 2013, Accepted: December 4, 2013

Abstract: Basic Local Alignment Search Tool (BLAST) is one of the most popular sequence alignment tools. BLAST consists of preprocessing, seeding, ungapped extension, gapped extension and traceback process. To accelerate BLAST, many hardware accelerators have been proposed. However, their acceleration target is mainly the gapped extension, and other parts are still realized as software that runs on a host machine. In this paper, we propose an accelerator for BLAST, which realizes all processing parts including the preprocessing and the traceback part as hardware. It could avoid the unbalanced processing speed between software and hardware. We also propose two performance improvement techniques for the gapped extension block. An inexpensive FPGA (field programmable gate array) implementation shows that our hardware accelerator performs 791 times faster than a software BLAST, with reasonable hardware cost. Moreover, by applying the performance improvement techniques, the performance of the accelerator becomes more than 840 times faster than the software BLAST.

Keywords: FPGA, BLASTP, Smith-Waterman, sequence alignment

1. まえがき

シーケンスアライメントとはタンパク質（またはDNA）配列データベース（DB）内のシーケンスと検索対象となるタンパク質（またはDNA）配列（クエリシーケンス）を比較し，配列どうしの類似部分検索を行うものである．シーケンスアライメントプログラムには，BLAST [1], [2], [3],

[1] 会津大学大学院コンピュータ理工学研究科，福島
Graduate School of Computer Science and Engineering, The University of Aizu, Aizuwakamatsu, Fukushima 965–8580 Japan
[a)] miyazaki@u-aizu.ac.jp

本論文の内容は2013年2月の平成24年度第4回情報処理学会東北支部研究会にて報告され，支部長により情報処理学会論文誌ジャーナルへの掲載が推薦された論文である．

FASTA [4], HMMER [5] などが存在する．その中で BLAST（Basic Local Alignment Search Tool）は最も有名なシーケンスアライメントツールの 1 つである．シーケンスアライメントは，生物学上の進化や遺伝子系図を調べるうえで重要であることから，バイオインフォマティクス分野では欠かせない情報となっている．そのため，BLAST の専用ハードウェアによる高速化が多く提案されている [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22]．しかしながら，それらの多くは，BLAST 処理の一部をハードウェア化するものであり，BLAST 処理全体を実現するには，相変わらず他の処理をホストマシン上でソフトウェアにより実現する必要がある [14]．そのため，一部処理だけが高速化できても，ホストマシンと専用ハードウェア間で処理速度のアンバランスが生じ，処理全体を高速化できない可能性があった．本研究では，BLAST アルゴリズム全体をハードウェア化することにより，上記不具合を解消し，BLAST 処理全体の高速化を図る．

2. BLAST アルゴリズム

BLAST アルゴリズムは，基本的な処理の流れは同じであるが，アライメント対象により，様々な種類が存在する．ここでは，利用の拡大が期待されるタンパク質配列のシーケンスアライメントを対象とした BLASTP [13], [15], [19] を考える．BLASTP アルゴリズムは，前処理，seeding（ステップ 1），ungapped extension（ステップ 2），gapped extension（ステップ 3）の 3 ステップと traceback 処理からなる [1]．以下に，アルゴリズムの各処理について詳しく説明する．

2.1 前処理

前処理では，まず，クエリシーケンスを k 文字（通常，タンパク質配列では $k = 3$，本論文でも $k = 3$ を用いる）のクエリワードに分割する（**図 1** 左図）．次に，そのクエリワードの 3 文字を他のアミノ酸の 20 文字と 1 文字ずつ比較し，スコアの合計を計算する．ここで，スコアとは，比較する 2 つの配列の類似度を数値化したものであり，文字ペアごとに，置換行列（Substitution Matrix）[1] を参照して得る．置換行列とは，タンパク質を構成する 20 種のアミノ酸どうしの類似度を数値化したものであり，一般に 20×20 の行列形式で表現される．置換行列は複数存在するが，本論文では，BLOSUM50 と呼ぶ置換行列を使用する．3 文字のスコアの合計が閾値 T（通常 $T = 12$）以上となったワードが隣接ワード（Neighborhood word）となる（図 1 右図）．生成された隣接ワードが次の seeding 処理で使用される．

2.2 ステップ 1：Seeding

ステップ 1 の seeding では，前処理で生成した隣接ワードと DB シーケンスを比較し，隣接ワードと完全に一致する部分を探す．この一致した部分は seed と呼ばれ次のステップ 2 で使用される．**図 2** では，DB シーケンスとクエリシーケンス上の TAL が seed となる．

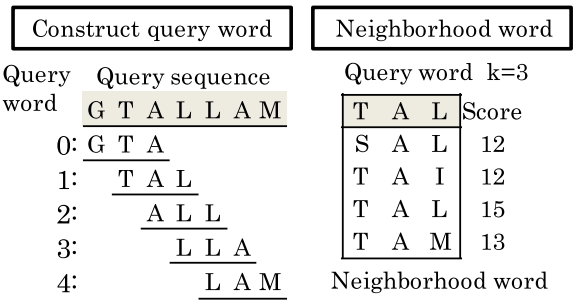


図 1 クエリワード作成と隣接ワードの作成例
Fig. 1 Example of query and neighborhood word creation.

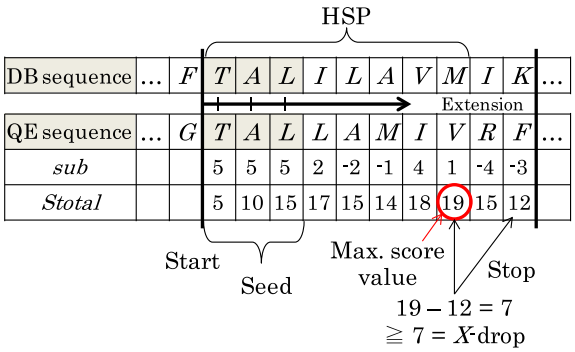


図 2 Seeding と ungapped extension の例
Fig. 2 Example of seeding and ungapped extension step.

2.3 ステップ 2：Ungapped extension

ステップ 2，すなわち ungapped extension ではステップ 1 で見つかった seed を拡張開始点とし，拡張を行う．seed から，DB，クエリシーケンスを拡張する場合，各文字を比較しスコアの計算を行う．図 2 で *sub* とは，対応する文字ペアが持つ類似度であり，前記置換行列を参照して得られる文字ペアごとに一意に定まる値である．また，*Stotal* とは，seed の先頭から，その時点までの *sub* 値の合計である．その *Stotal* 値が，それまでの最大値から閾値 X（本論文では，一般的な $X = 7$ を使用）下がるまで 1 文字ずつ拡張を行う．拡張されたシーケンスペアのうち最大スコアが閾値 S（本論文では，$S = 11$）以上となる組合せを HSP（High-scoring segment pair）と呼び，次のステップの入力となる．図 2 では，シーケンスペア TALILAVM と TALLAMIV が HSP となる．

2.4 ステップ 3：Gapped extension

ステップ 3 の gapped extension では Smith-Waterman アルゴリズム [1] という動的計画法に基づく手法を使用している．Smith-Waterman アルゴリズムは下式を使用する．

Initialization:

$$F(m,0)=0 \text{ where } 0 \le m \le M$$

$$F(0,n)=0 \text{ where } 0 \le n \le N$$

$$F(m,n)=\max \begin{cases} 0 \\ F(m-1,n-1)+sub(x_m,y_n) \\ F(m-1,n)-d \\ F(m,n-1)-d \end{cases} \quad (1)$$

ここで，$sub(x_m,y_n)$ と d は定数である．$sub(x_m,y_n)$ は，対応する文字ペアのスコアであり，置換行列を参照して得られる値である．また，d は，“ギャップペナルティ” と呼ぶ定数であり，本論文では，$d=8$ とする．

本アルゴリズムでは，計算処理に 2 次元データ行列（以下，スコア行列と呼ぶ）を用いる．**図 3** は，HSP をスコア行列の行と列に対応させ，各行列要素（式 (1) の $F(m,n)$，以下，セルと呼ぶ）のスコア計算を行った結果である．スコア行列の，1 行目および 1 列目は初期値として 0 が割り当てられる．1 つのセルは，式 (1) 右辺の斜め上 $F(m-1,n-1)$，上 $F(m-1,n)$，そして左 $F(m,n-1)$ のセルのスコアを使用して計算された値と，数値 “0”，の 4 つの値を比較し，その最大値をスコアとして該当セルに保持する．図 3 中の矢印は，どの位置のスコアを使用したのか（以下，経路と呼ぶ）を示している．

2.5 Traceback 処理

traceback 処理では gapped extension 処理で求めたスコア行列上で，最も高いスコア（図 3 では 20）を持つセルからスコアが 0 のセルまでの経路をたどり，最適なシーケンスを出力する．図 3 の例では，“TALILAVM” と “TAL-LAMI” が最終的に得られる最適なシーケンスアラインメントである．ここで，アラインメント中の “-” はギャップと呼ばれ，文字列の曖昧性（ずれ）を示す．どの程度連続したギャップを許容するかは，式 (1) 中のギャップペナルティ “d” の値により制御できる．このように，BLAST では，2 つの文字列シーケンスの完全マッチングでなく，曖昧性を包含したマッチングを可能とする．

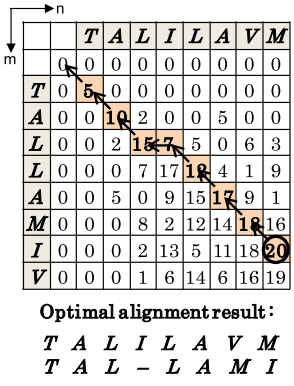


図 3 Gapped extension と traceback 処理の例

Fig. 3 Example of gapped extension step and traceback process.

3. 提案回路の全体構成

本研究では，前述した BLAST 処理全体のハードウェア化を行った．**図 4** に提案回路の全体構成を示す．

提案回路は，Block 1 から Block 4 の 4 つの部分からなる．各部は，前述した隣接ワードを求める前処理（Block 1），ステップ 1，2，3（Block 2〜4）に対応している．Block 4 には，gapped extension 処理部に加え，traceback 処理部も含まれる．前処理部では，入力されたクエリシーケンスを用いて隣接ワードメモリの内容を生成する．この隣接ワードメモリは，Block 2 の処理で，繰り返し参照される．Block 2 では，shifter を用いて，DB メモリ内の DB シーケンスを 3 文字に分割し DB ワードの生成を行う．DB ワードはアドレス生成部に送られ，隣接ワードメモリの読み出しアドレスが生成される．読み出しアドレスによって位置情報が出力された場合，同一文字の一致点（seed）が見つかったことを示す．Block 3 の ungapped extension 処理では，パイプライン処理を容易に導入できるように，文字列を左右方向に拡張するのではなく，文献 [21] が提案している 1 方向へのみ拡張する方式とした．Block 3 はアドレス生成部，DB，クエリシーケンスのメモリブロックとスコア計算部から構成されている．アドレス生成部では，seed の点を拡張開始点とし，読み出しアドレスを 1 ずつ加算していく．スコア計算部では，DB，クエリシーケンスのメモリから出力された文字ペアのスコアを，置換行列を参照して得るとともに，そのスコア値を足し込み，合計を求める．さらに，それまでに求まったスコアの最大値から，今求めたスコアの合計を減じ，その減少値があらかじめ定めた閾値 S を超えたら拡張を終了させる制御を行う．Block 3 の処理で求まった HSP は時間調整用の FIFO を介して，最終処理部である Block 4 へ送られる．Block 4 では，Smith-Waterman アルゴリズムに基づいた計算が行われ，計算終了後，最適

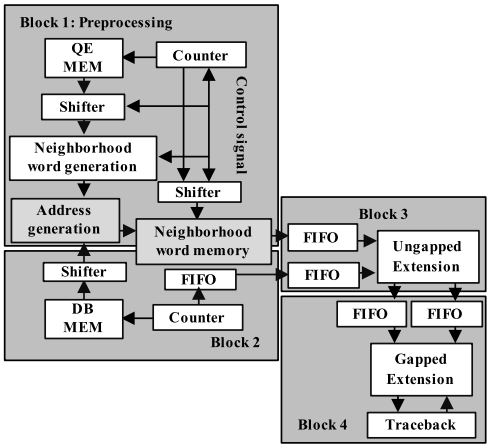


図 4 BLAST 処理を実行する提案回路の全体構成

Fig. 4 Overview of the proposed hardware architecture for BLAST.

なシーケンスペアを出力する．traceback 処理終了後，接続された FIFO から gapped extension 処理部へ，次の HSP が送られる．上述した処理を，FIFO 内のすべての HSP に対して行う．

Block 2〜4 の処理はパイプラン化しており，Block 間のデータ転送は，遅滞なく行うことができる．これは，Block 1 の前処理部と Block 4 内の traceback 処理をハードウェア化し，ソフトウェア（ホストマシン）とハードウェア間での処理速度のアンバランスを解消することにより実現できたものである．

Block 2〜3 の動作については，文献 [13] の構成を参考にしたが，パイプライン処理が適用可能なように変更を加えている．また，従来実装では，Block 4 の計算部については，Block 3 から送られる文字列の長さにかかわらず，内部で使用されている 1 次元接続された PE（Processing Element，詳細は後述）数に依存したクロック数が必要となっていた [9]．ここでは，Block 3 から送られる文字列の先頭および末尾に，始点・終点を表す特殊文字を挿入することより，PE 数に依存せず，終点文字が入力されると，瞬時に計算終了が判断でき，また，始点文字到来とともに，次の文字列に対する計算を開始できるように工夫した．

4. 各処理部の詳細

従来，BLAST のハードウェア化は，3 つのステップのうち，最も計算量が大きいステップ 3，すなわち gapped extension を中心に行われてきた [9], [10], [12], [16]．そのため，前処理である隣接ワードの生成や traceback 処理は，ホストマシン上のソフトウェアで行う必要があった．しかし，前処理部で生成される隣接ワードは膨大であるため，ホストマシンで生成された隣接ワードをハードウェアへ転送するには時間がかかる [14]．また，gapped extension 処理後に実行される traceback 処理は，ローカルメモリに保存されたデータを使用するため，traceback 処理が終了しなければ，次の入力データに対して gapped extension 処理を行うことができず，ホストマシンとハードウェア間で処理速度のアンバランスが生じる可能性があった．本論文では，前述した前処理部と traceback 処理を含む BLAST アルゴリズム全体をハードウェア化することを提案する．本章では，新たに提案する前処理部，および，traceback 処理のハードウェア実装を中心に述べる．

ここでは，BLASTP（タンパク質配列のシーケンスアライメントツール）[19] を実装対象とし，クエリワード長 3 ($k = 3$)，各閾値を，$T = 12$，$X = 7$，$S = 11$ とする．まず，前処理部について説明する．**図 5** に前処理を行う提案回路を示す．前処理部はクエリ生成部，隣接ワード生成部，アドレス生成部で構成されている．まず，クエリワード生成部では，クエリシーケンスを 3 文字のワードに分割し，クエリワードを生成する．次に，隣接ワード生成部では，生

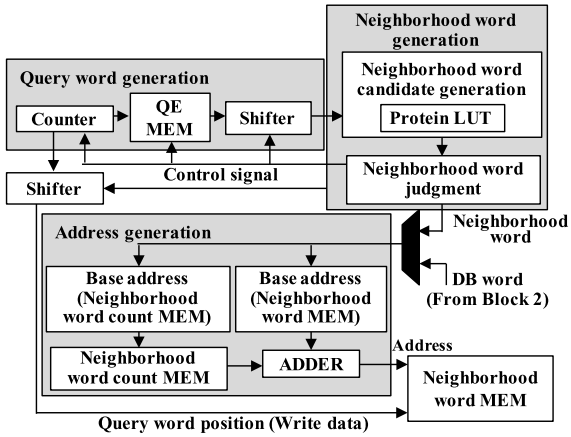
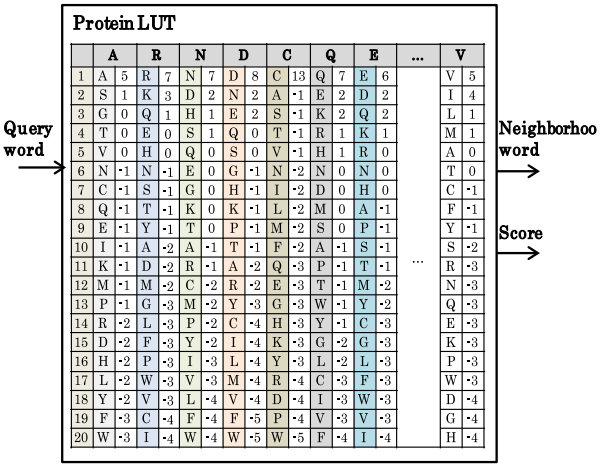


図 5 前処理部の提案回路
Fig. 5 Overview of the preprocessing block.



	A		R		N		D		C		Q		E		...	V	
1	A	5	R	7	N	7	D	8	C	13	Q	7	E	6		V	5
2	S	1	K	3	D	2	N	2	A	-1	E	2	D	2		I	4
3	G	0	Q	1	H	1	E	2	S	-1	K	2	Q	2		L	1
4	T	0	E	0	S	1	Q	0	T	-1	R	1	K	1		M	1
5	V	0	H	0	Q	0	S	0	V	-1	H	1	R	0		A	0
6	N	-1	N	-1	E	0	G	-1	N	-2	N	0	N	0		T	0
7	C	-1	S	-1	G	0	H	-1	I	-2	D	0	H	0		C	-1
8	Q	-1	T	-1	K	0	K	-1	L	-2	M	0	A	-1		F	-1
9	E	-1	Y	-1	T	0	P	-1	M	-2	S	0	P	-1		Y	-1
10	I	-1	A	-2	A	-1	T	-1	F	-2	A	-1	S	-1	...	S	-2
11	K	-1	D	-2	R	-1	A	-2	Q	-3	P	-1	T	-1		R	-3
12	M	-1	M	-2	C	-2	R	-2	E	-3	T	-1	M	-2		N	-3
13	P	-1	G	-3	M	-2	Y	-3	G	-3	W	-1	Y	-2		Q	-3
14	R	-2	L	-3	P	-2	C	-4	H	-3	Y	-1	C	-3		E	-3
15	D	-2	F	-3	Y	-2	I	-4	K	-3	G	-2	G	-3		K	-3
16	H	-2	P	-3	I	-3	L	-4	Y	-3	L	-2	L	-3		P	-3
17	L	-2	W	-3	V	-3	M	-4	R	-4	C	-3	F	-3		W	-3
18	Y	-2	V	-3	L	-4	V	-4	D	-4	I	-3	W	-3		D	-4
19	F	-3	C	-4	F	-4	F	-5	P	-4	V	-3	V	-3		G	-4
20	W	-3	I	-4	W	-4	W	-5	W	-5	F	-4	I	-4		H	-4

図 6 Protein LUT
Fig. 6 Protein LUT.

成された各クエリワードに対して，隣接ワードの生成を行う．隣接ワードを得るには，前述したように，クエリワードの各文字に対して，アミノ酸 20 種類の対応があるため，$20 \times 20 \times 20 = 8{,}000$ 種類のワードに対して，スコアを計算し，その値が閾値 T 以上のものを選出する処理が必要となる．本処理を効率化するために，置換行列を列ごとに，降順にソートしたテーブルを持つタンパク質 Lookup Table（Protein LUT）を設ける．**図 6** に Protein LUT の一例を示す．図に示すように，たとえばクエリワード “ARN” に対する最初の隣接ワード候補は，図 6 の LUT の最上位にある “ARN” であり，そのスコアは，$5+7+7 = 19$ であることが即座に分かる．また，そのスコアは閾値 $T = 12$ を超えているので，ARN は ARN の隣接ワードとなる．次の候補は，A の列と R の列は 1 番目，N の列は 2 番目のアミノ酸の組合せ ARD であり，そのスコアは $5+7+2 = 14$ であるから，これも閾値 T を超えるので隣接ワードとなる．同様に N の列を下方に進めていくと，ARH, ARS, ..., ART は，隣接ワードとして出力される．次の ARA は，そのスコアが $5+7-1 = 11$ となり閾値 T より小さくなる

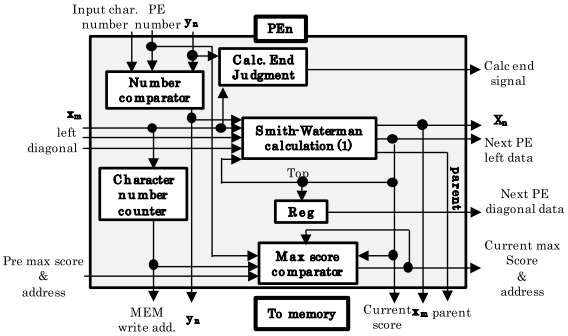


図 7 PE の内部構造
Fig. 7 Structure of PE.

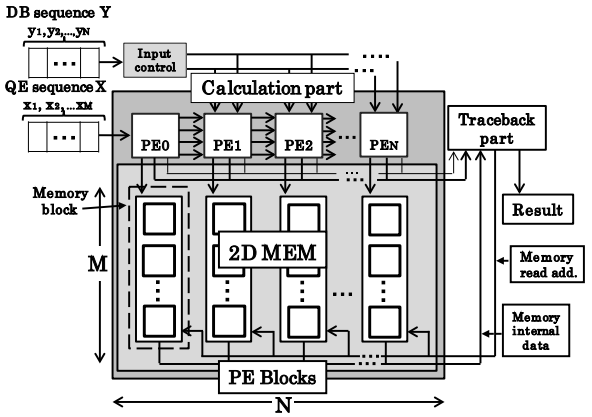


図 8 gapped extension 処理と traceback 処理を実行する提案回路
Fig. 8 Proposed hardware for gapped extension and traceback processes.

ため隣接ワードではない．また，N の列の A より下方のアミノ酸を組み合わせても，閾値 T を超えることはないので，N の列の候補変更は，即座にやめ，R の列のアミノ酸を 2 番目以降に進め，上記と同様に閾値 T よりスコアが小さくなったところでやめるという操作を行う．上述したように，Protein LUT を使用すれば，1 つのクエリシーケンスに対して，毎回 8,000 通りの組合せすべてのスコアを逐次調べるのに比べ，入力クエリワードに対する隣接ワードの生成を効率的に行うことができる．ステップ 1 以降で必要となるデータは隣接ワードがどのクエリワードに対して生成されたかを示す位置情報である．当該位置情報を保持するために，隣接ワードメモリを用意する．アドレス生成部では，クエリワードに対し生成された隣接ワードを使用し，隣接ワードメモリの書き込みアドレスを生成する．ここで，メモリに保持する隣接ワードの位置情報の数は，クエリシーケンスの長さとアミノ酸の種類に依存して増加する．1 つの隣接ワードに対して格納できる位置情報の最大数を超えた場合は，未使用のメモリ領域を検出し，そこに位置情報を保存していく．隣接ワードメモリはステップ 1 の処理で使用する．よって，すべての隣接ワードの位置情報がメモリに格納された後，ステップ 1 の処理が開始される．ステップ 1 では，DB シーケンスの 3 文字のワードが入力され，アドレス生成部では，隣接ワードの読み出しアドレスが出力される．隣接ワードから，データが出力された場合，seed が見つかったこととなる．このように，前処理部のハードウェア化を行うことで，ソフトウェアとハードウェア間でデータ転送を行うことなく処理を進めることが可能となる．

次に，traceback 処理について説明する．**図 7** に，Processing Element（PE）の内部構造，**図 8** に gapped extension 処理と traceback 処理を実行する提案回路を示す．gapped extension 部内の計算には，Processing Element（PE）と呼ぶ簡単な処理を行うプロセッサを複数使用する．図 8 のように複数の PE どうしを 1 次元接続して構成する PE アレイで並列処理を可能とし，ソフトウェア処理に比べ高速化を実現する．PE アレイを用いた処理の高速化は，文献 [6], [20] にもみられるが，traceback 処理部のハードウェア化を前提としていないため，PE アレイのデータ入出力において，本提案回路とは異なり，複雑な周辺回路を要求する．各 PE にはローカルメモリを接続し，計算結果は，ホストマシンに転送せずに，当該メモリに格納する．ローカルメモリは $M \times N$（$M >$ クエリシーケンス長，$N >$ DB シーケンス長）の 2 次元メモリ（2D MEM）で各計算結果を図 3 で示したスコア行列に対応する位置に格納できる．ローカルメモリに格納される各計算結果は，4 つ組の情報（行列に対応したクエリ，DB シーケンス中の各 1 文字，どの位置のスコアを使用したかを示す parent（図 3 の対応矢印の終点の座標値），計算によって求められたスコア）からなる．PE アレイの計算が終了すると，クエリの終端文字を保持する PE から，終了信号とともに，最大スコアと当該最大スコアが格納されているメモリアドレスが traceback 処理部に送られ，ただちに traceback 処理が開始される．

traceback 処理部では，PE から届いた最大スコアのメモリアドレスを最初の読み出しアドレスとして，ローカルメモリにアクセスする．メモリから出力された四つ組データは traceback 処理部に送られる．traceback 処理部は，四つ組データ中の parent 値を次のメモリ読み出しアドレスとして，再びローカルメモリにアクセスする．一方，クエリおよび DB の文字データは，そのまま出力する．本操作を，ローカルメモリから読み出した四つ組データのスコア値が 0 になるまで繰り返す．上記の操作において，traceback 処理部から出力されたデータが最適なシーケンスとなる．

このように，traceback 処理部をハードウェア化することで，Block 内部のデータ転送をスムーズに行うことができるようになる．

5. 高速化手法

ここでは，前章で述べた BLAST 処理全体を実行するハードウェアで，最も重い処理である gapped extension 処

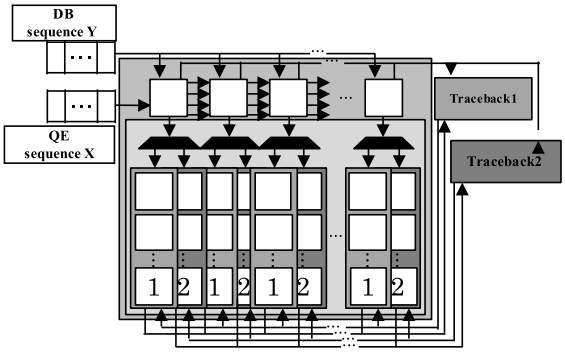


図 9 Dual memory を使用した gapped extension 処理と traceback 処理を実行する提案回路

Fig. 9 Gapped extension and traceback parts with the dual memory.

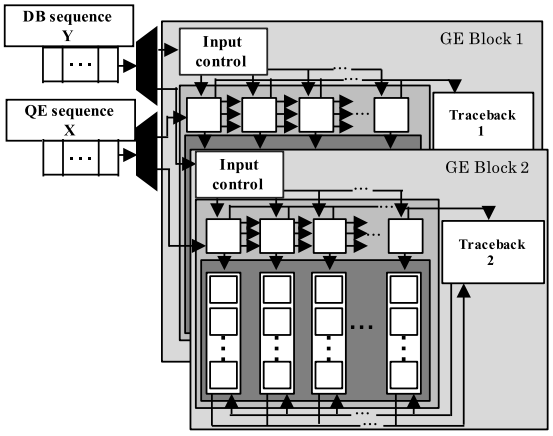


図 10 gapped extension block の並列化

Fig. 10 Parallelization of the gapped extension block.

理をさらに高速化する手法を提案する．

5.1 Dual memory

Block 4（gapped extension block）は 2 次元メモリと traceback 処理部で構成されている．traceback 処理は 2 次元メモリに格納された計算結果を使用するため，PE アレイでの計算がすべて終了するまで開始することはできない．これを改善するために，2 次元メモリを 2 面設ける Dual memory 構造を用いる PE を提案する．

図 9 に Dual memory を使用した gapped extension 処理と traceback 処理を実行する提案回路を示す．Dual memory はそれぞれの PE にデマルチプレクサを介して接続されており，制御信号にメモリへの切替えが行われ，計算結果が保存される．Dual memory を用いた回路の動作の流れを以下に示す．ここで，Memory 1 とは，2 面設けたメモリのうち，1 面目のメモリ，Memory 2 とは 2 面目のメモリを指す．

1. 最初に入力された HSP に対し Smith-Waterman アルゴリズムに基づいた計算処理を行う．計算結果を Memory 1 に保存する．HSP に対して PE でのすべての計算が終了すると，PE から Traceback 1 に traceback 開始データが送られる．
2. 次に入力された HSP に対し計算処理を行い，計算結果を Memory 2 に保存する．本計算処理と同時に Traceback 1 処理部では Memory 1 に保存されているデータに対し，traceback 処理を行う．PE でのすべての処理が終了すると，PE から Traceback 2 処理部に traceback 開始データが送られる．
3. 次の HSP に対して計算処理を行い，計算結果を Memory 1 に保存する．本計算処理と同時に Traceback 2 処理部では Memory 2 に保存されているデータを使用し，traceback 処理を行う．PE でのすべての処理が終了すると，PE から Traceback 1 処理部に traceback 開始データが送られる．
4. 処理 2，3 をすべての HSP に対して繰り返す．

Dual memory を使用することで，PE での計算処理と，traceback 処理を同時に実行できる．これにより，さらなる高速化が可能となる．

5.2 Gapped extension block の並列化

2 つ目に，gapped extension block（GE Block）の並列化を提案する．Dual memory を使用した場合，各シーケンスの入力に時間がかかる可能性がある．これを解消するために，GE Block の全体の並列化を行う．

図 10 に GE Block を並列化した提案回路を示す．DB，クエリが保持されている FIFO はそれぞれ 1 つであるため，2 つの GE Block へ入力はマルチプレクサを介して行われる．処理手順は下記のとおりである．

1. GE Block 1 に対して，FIFO から HSP データが入力される．入力後，GE Block 2 へ次の HSP データが入力される．入力と同時に，GE Block 1 では，計算処理，traceback 処理が行われる．
2. GE Block 2 への入力が終了後，GE Block 1 での処理が終了している場合，ただちに次の HSP データが GE Block 1 へ入力される．同時に，GE Block 2 では計算処理，traceback 処理が行われる．

1, 2 をすべての HSP に対して行うことで，Dual memory を用いた場合よりも高速化が期待できる．

6. 実験および評価

FPGA を用いて提案回路の実装を行った．また，Dual memory を用いた回路，GE Block を並列化した回路の実装評価も行った．今回は，動作確認，回路規模の確認のため DB，クエリシーケンス長ともに 100 であることを前提に回路の実装を行った．使用した FPGA は，評価ボード DE2-115 [23] に実装されている Altera 社 Cyclone-IV E EP4CE115F29C7 である．回路実装には同社の設計ツール QuartusII 10.1sp1 を使用した．

表 1 回路全体の FPGA 回路規模
Table 1 FPGA logic utilization of the proposed hardware.

リソース	使用量	使用率
ロジックエレメント数	41,142	36%
レジスタ数	3,656	3%
メモリビット数	653,056	16%

表 2 高速化手法（Dual memory）を用いた全体回路の FPGA 回路規模
Table 2 FPGA logic utilization of the proposed hardware when the acceleration method (Dual memory) is introduced.

リソース	使用量	使用率
ロジックエレメント数	44,247	39%
レジスタ数	3,914	3%
メモリビット数	745,056	19%

表 3 高速化手法（GE Block の並列化）を用いた全体回路の FPGA 回路規模
Table 3 FPGA logic utilization of the proposed hardware when the acceleration method (Parallelized GE Block) is introduced.

リソース	使用量	使用率
ロジックエレメント数	79,303	69%
レジスタ数	6,807	6%
メモリビット数	745,056	19%

表 4 ソフトウェア BLAST と提案回路の速度比較
Table 4 Performance comparison between the proposed hardware and software BLAST.

	実行時間	高速化率
ソフトウエア BLAST	135.0 ms	1
提案回路	0.17 ms	791
Dual memory	0.16 ms	844
GE Block の並列化	0.15 ms	900

表 1 は図 4 に示した回路全体の実装結果である．また，**表 2** は図 9 の回路を含む Dual memory を用いた回路全体の実装結果である．さらに，**表 3** は GE Block を並列化した回路（図 10）の実装結果を示している．これらの結果から分かるように，提案回路と Dual memory を使用した回路の回路規模は，比較的小さい．そのため今後長いシーケンス長に対応するために提案回路全体の拡張を行うことは容易である．一方，GE Block を並列化した回路は，提案回路に比べ約 2 倍の回路規模となっている．**表 4** はソフトウェア BLAST，提案回路および高速化手法を使用した回路の実行時間を比較したものである．

評価に用いたクエリシーケンス，DB シーケンス長はともに 100 である．提案回路の最大動作周波数は 55.11 MHz，上記シーケンスを処理するのに要したクロック数は 9,383 であり，実行時間は 0.17 ms であった．比較対象として，ソフトウェア BLAST（Local BLAST 2.2.25）を使用し，市販 PC 上で動作させた．使用した PC は，本評価実験を行った時点で，FPGA 評価ボード DE2-115 [23] と同程度の

表 5 各処理部のソフトウェア BLAST と提案回路の比較
Table 5 Performance improving ratio of each hardware block compared to the corresponding processing part in the software BLAST.

	提案回路	ソフトウエア	高速化率
前処理部	0.12 ms	32.4 ms	270
seeding	30 us	50.0 ms	1667
UE & GE	490 us	52.7 ms	1075

市場価格の Intel Core 2 Duo（4 GB メモリ）を搭載したマシンである．その実行結果は，同一入力データに対して，135 ms であった．よって，表 4 に示すように，提案回路はソフトウェアに比べて 791 倍の高速化を実現している．

また，Dual memory を用いた回路の最大動作周波数は 53.48 MHz，クロック数は 8,806 であり，実行時間は 0.16 ms であった．さらに，GE Block の並列化した回路の場合，動作周波数は 55.82 MHz，クロック数は 8212 であったことから，各回路の実行時間は 0.16 ms，0.15 ms となり，ソフトウェアに比べて 844 倍，900 倍の高速化を実現した．

今回，同程度のハードウェアコストで，FPGA とソフトウェアを比較していることから，上記高速化は，そのままコスト性能比ととらえることもできる．また，今回使用した FPGA は比較的安価であり，実現できる最大回路規模も大きくないが，提案回路は十分搭載できた．よって，コストをかけずに，提案回路を複数並列に動作させることによる高速化も容易である．一方，CPU の性能価格比は，線形でなく，高速化するためには，一般に指数的なコストがかかる．本事項を考慮すると，我々の提案回路はコスト性能比という点で有利であるといえる．

最後に，BLAST 各部の実行時間の内訳を示す．今回比較に用いたソフトウェア BLAST は，各処理部が実装上きれいに分かれておらず，入力データにも各処理部の実行時間が依存する．そこで，文献 [22] を参考に，その内訳を算出した．文献 [22] によれば，前処理部，seeding，ungapped extension，gapped extension（traceback 処理を含む）の実行時間の割合は，それぞれ全体の 24%，37%，15%，24%である．なお，文献 [22] で提案している gapped extension プログラムもパイプライン処理を導入している．そのため，上記実行時間の割合は，我々の実効時間の内訳に用いても，大きな誤差は生まないと判断し，当該実効時間内を使用することとした．本評価でのソフトウェア BLAST の実行時間が 135 ms であったことから，それぞれの処理の実行時間は，前記の比率から 32.4 ms，50.0 ms，20.3 ms，32.4 ms となる．各ステップの実行時間の比較を**表 5** に示す．表 5 より，前処理部では，270 倍，seeding，ungapped & gapped extension（UE & GE）部では 1,000 倍以上の高速化を実現したこととなる．

また，我々は，本論文で用いた Smith-Waterman アルゴリズムを実行する回路を単独で評価し，文献 [9], [25] の回路

に比べて 2 倍程度の速度向上を得ていることを示した [24]．ここで，本提案の gapped extension 回路では，3 章で述べたように前段の ungapped extension 回路の出力文字の先頭と末尾に始点・終点を表す特殊文字を挿入することで，文献 [24] のそれに比べ，PE 数に依存しないより効率的な処理を行えるようになっている．一方，文献 [9] および [25] では，実現した回路は，それぞれソフトウェア実装に比べ，数百倍および約 330 倍であったと報告している．よって，本提案の gapped extension 回路は，ソフトウェア BLAST に比べ，1,000 倍程度の高速化が期待でき，それは前記評価結果とも一致する．

今回提案した回路の中で最もクロック数を要したのは，前処理部を実行する Block 1 の回路であった．この原因は，今回評価に使用した DB シーケンス長が 100 と比較的短かったことがあげられる．すなわち，今回は，DB シーケンス長およびクエリシーケンス長ともに同じ 100 であり，検索時間よりも隣接ワードメモリ内部のデータ生成に相対的に時間を要したのが原因である．一般に，クエリシーケンス長に対し，DB シーケンス長の方が長い．また，複数の DB シーケンスが検索対象となる．それゆえ，実際の処理では，前処理部の実行時間はボトルネックにはならない．ただ，次ステップ以降の処理を，5 章で述べたように，並列化し，異なるデータを同時に処理する場合，遅滞なく入力データを供給する必要が出てくることから，前処理部をハードウェア化することは意味がある．また，今回の評価では，提案回路と高速化手法を用いた回路では，大幅な性能改善はみられなかった．これも，検索対象として使用した DB シーケンス長が短いことが主な要因である．

5 章で述べた 2 つの高速化手法は，ともにスループットを改善するものであり，データ量が増えれば，導入効果は大きいと考える．

7. 考察

前述したように，我々は，コスト性能比に優れた構成を志向し，廉価な FPGA で実装評価を行った．ここでは，FPGA ボードとホストマシンとして PC を用いた従来法の実装に関して考察する．

今，前処理部，seeding 処理，traceback 処理を PC で，ungapped と gapped extension 処理を FPGA ボードで実現する場合を考える．クエリワード 1 つに対し，隣接ワードは，300〜500 個程度存在する．よって，たとえば，クエリシーケンス長とクエリワード長 k をそれぞれ 100，$k = 3$ とすると，$100 - k + 1$（クエリワード数）$\times 500$（隣接ワード数）$\times 4$（隣接ワードのクエリシーケンスおよび DB シーケンス上の始点位置情報，各 2 B）$= 196$ KB となる．ここで，FPGA ボードを PC に接続して使用する場合，PCI express を用いるのが一般的である．PCI express Gen1 では，1 レーンあたりの実効通信速度のピーク値は，単方向，約 250 MB/s ある．よって，通信ボトルネックは生じない．しかし，表 5 から分かるように，ソフトウェア処理においては，前処理と seeding 処理の実行時間が 83.4 ms，ungapped と gapped extension 処理のそれが 52.7 ms であることから，その比は 1.6 倍以上ある．このことから，ungapped と gapped extension 処理のみをハードウェア化しても，処理全体のボトルネックはソフトウェア処理にある．よって，BLAST 処理全体をハードウェア化する本論文のアプローチは，全体の処理が高速化するという意味で重要である．

また，traceback 処理を PC で行う場合，gapped extension（Smith-Waterman アルゴリズム）を実行した Block 4 に内在するスコア行列のデータを，PC に転送する必要がある．今，Black 4 に入力された HSP の各長さを n とすると，スコア行列のデータ量は，$n^2 \times 4$ B（1 データ 4 B と仮定）となる．ここで，表 5 から 400 us ごとに当該データを PC に送るとし，$n = 25$ とすると，$(25^2 \times 4)/(400 \times 10^{-6}) = 6.25$ MB となる．本データを PCI express インタフェースにより，PC に送ること自体は特に問題とならないが，5 章で述べた並列化による高速化手法を導入する場合，PC に転送しなければならないスコア行列データ量は，並列度に対して線形に増加する．単純計算では，250 MB/6.25 MB = 40 並列まで行っても，PCI express の通信ボトルネックにはならないが，実際は，ソフトウェア処理との同期や相互にデータを交換するためのセットアップに要する時間が必要となるため，そこまでの並列化は望めない．一方，本論文が提案するように，traceback 処理もハードウェア化し，並列化に際しては，treceback 処理部も並列化可能とすれば，他の処理とのインタフェースを大きく変更することなく，独立性を保ったまま，単純に並列度に応じた高速化が実現できる．

Dual memory を使用した回路と，並列化を行った回路では Block 4 で用意されているメモリ数には違いはない．しかしながら，並列化を行った場合は PE 数が 2 倍となるため，PE を多く使用している場合にはハードウェアコストが大幅に増加する．一方，性能に関しては，ソフトウェアと比較し Dual memory が 844 倍，並列化回路が 900 倍となった．このことから，リソースに制限がある場合には Dual memory を使用し，速度を重視する場合には並列化回路を導入するのがよいと考える．

また，今回使用した比較的短い 100 のシーケンス長に対して使用した総メモリ量は，5 章の高速化手法を適用しても 745 KB であった（表 2 および表 3 参照）．しかし，シーケンス長を一般的な長さ 500〜1,000 にした場合では，FPGA 内のブロックメモリでは容量が不足する可能性が高い．現実装では，図 4 で示した Block 1 のクエリメモリおよび Block 2 の DB メモリ内容と同一内容を Block 3 でも保持している．それらを共有化すると，上述した総メモリ量は，半減できる．さらに，外部メモリを FPGA に接続し，必要なデータのみを FPGA 内のブロックメモリに動

的にロードする機構を導入することにより，一般的な長さのシーケンス長にも対応できると考える．なお，各 Block の回路が参照するべきデータはあらかじめ分かるため，外部メモリからブロックメモリ内に動的にロードすべきデータを同定すること，およびそのスケジューリングは，比較的単純なハードウェア機構で実現できる．

8. むすび

BLAST 処理全体を実行するハードウェアアーキテクチャを提案した．FPGA 実装した提案回路が，FPGA 実装環境と同等コストの PC で実行したソフトウェア BLAST よりも 790 倍以上高速化できることを示した．また，性能向上策として Dual memory を使用することで，gapped extension block の計算処理と traceback 処理を並列実行させた場合，841 倍，gapped extension block 全体を並列化した場合，900 倍の高速化が実現できることを示した．提案回路はコスト性能比で優れている．

ungapped extension 処理から出力される HSP（High scoring segment pair）の個数を減らせれば，後処理である gapped extension 処理を行う回数が減り，より効率化できる．上記目的のために，ソフトウェア実装では ungapped extension 処理において，一定以上のシーケンス長を持つ HSP のみを出力する機構として Two-hit 法 [15], [19] がよく知られている．当該法の導入が今後の課題の 1 つである．

参考文献

[1] Korf, I., Yandell, M. and Bedell, J.: *BLAST*, pp.3–95, O'REILLY (2003).

[2] Altschul, S.F., Gish, W., Miller, W., Myers, W. and Lipman, D.J.: *Basic Local Alignment Search Tool*, pp.403–410, Academic Press Limited (1990).

[3] Fassler, J. and Cooper, P.: NCBI: BLAST Help (2011), available from ⟨http://www.ncbi.nlm.nih.gov/books/NBK62051/pdf/blast_glossary.pdf⟩.

[4] Elofsson, A.: Lecture about BLAST and FASTA (2003), available from ⟨http://www.bioinfo.se/kurser/swell/blasta-fasta.shtml⟩.

[5] HHMI: HMMER biossequence analysis using profile hidden Markov models, available from ⟨http://hmmer.janelia.org/⟩.

[6] Jacob, A.C., Lancaster, J.M. Buhler, J.D. and Chamberlain, R.D.: FPGA Acceleration of Seeded Similarity Searching, *Bioinformatics High Performance Parallel Computer Architectures*, B. Schmidt (2010).

[7] Lavenier, D. and Nguyen, V.: Seed-Based Parallel Protein Sequence Comparison Combining Multithreading, GPU, and FPGA Technologies, *Bioinformatics High Performance Parallel Computer Architectures*, B. Schmidt (2010).

[8] Guo, X., Wang, H. and Devabhaktuni, V.: Design of a FPGA-Based Parallel Architecture for BLAST Algorithm with Multi-hits Detection, *IEEE 8th International Conference on Information Technology: New Generations* (*ITNG*), pp.689–694 (online), DOI: 10.1109/ITNG.2011.122 (2011).

[9] Benkrid, K., Liu, Y. and Benkrid, A.: A Highly Parameterized and Efficient FPGA-Based Skeleton for Pairwise Biological Sequence Alignment, *IEEE Trans. Very Large Scale Integration* (*VLSI*) *Systems*, Vol.17, No.4, pp.561–570 (2009).

[10] 山口佳樹，宮島洋介，丸山　勉，小長谷明彦：書き換え可能ハードウエアを用いた高速ホモロジー検索システム，情報処理学会論文誌（HSP），Vol.43, No.6, pp.196–205 (2002).

[11] Kasap, S., Benkrid, K. and Liu, Y.: Design and Implementation of an FPGA-based Core for GappedBLAST Sequence Alignment with the Tow-Hit Method, International Association of Engineers (2008), available from ⟨http://www.engineeringcharacters.com/issues_v16/issue_3/EL_16_3_25.pdf⟩.

[12] 福井　啓，藤田昌宏：FPGA を用いた Smith-Waterman Algorithm の高速化，信学技報，Vol.111, No.31, pp.67–72, 電子情報通信学会 (2011).

[13] Jacob, A., Lancaster, J. and Buhler, J.: Mercury BLASTP: Accelerating Protein Sequence Alignment, *ACM Trans. Reconfigurable Technology and System*, Vol.1, No.2 (online), DOI: 10.1145/1371579.1371581 (2008).

[14] Guo, X., Wang, H. and Devabhaktuni, V.: A Systolic Array-Based FPGA Parallel Architecture for the BLAST Algorithm, *International Scholarly Research Network ISRN Bioinformatics* (online), DOI: 10.5402/2012/195658 (2012).

[15] Jacob, A., Lancaster, J., Buhler, J. and Chamberlain, R.D.: FPGA-accelerated seed generation in Mercury BLASTP, *Proc. 15th IEEE Symposium on Field-Programmable Custom Computing Machines* (*FCCM*), pp.95–106 (2007).

[16] Yamaguchi, Y., Tsoi, H.K. and Luk, W.: FPGA-Based Smith-Waterman Algorithm: Analysis and Novel Design, *ARC*, LNC6578, pp.181–192, Springer (online), DOI: 10.1007/978-3-642-19475-7_20 (2011).

[17] Herbordt, M.C., Model, Gu, J.Y., Sukhwani, B. and VanCourt, T.: Single Pass, BLAST-Like, Approximate String Matching on FPGAs, *Proc. 14th IEEE Symposium on Field-Programmable Custom Computing Machines* (*FCCM*), pp.217–226 (2006).

[18] Herbordt, M.C., Model, J., Sukhwani, B., Gu, Y. and VanCourt, T.: Single Pass Streaming BLAST on FPGAs, *Parallel Computing*, Vol.33, pp.741–756 (online), DOI: 10.1016/j.parco.2007.09.003 (2007).

[19] Mahram, A. and Herbordt, M.C.: Fast and Accurate NCBI BLASTP: Acceleration with Multiphase FPGA-Based Prefiltering, *Proc. 24th ACM International Conference on Supercomputing* (*ICS'10*), pp.73–82 (2010).

[20] Masuno, S., Maruyama, T., Yamaguchi, Y. and Konagaya, A.: Multiple Sequence Alignment Based on Dynamic Programming Using FPGA, *IEICE Trans. Information and Systems*, Vol.E90-D, No.12, pp.1939–1946 (2007).

[21] Lancaster, J., Buhler, J. and Chamberlain, R.: Acceleration of Ungapped Extension in Mercury BLAST, *Microprocessors and Microsystems*, Vol.33, No.4, pp.281–289 (online), DOI: 10.1016/j.micpro.2009.02.007 (2009).

[22] Altschul, S.F., Madden, T.L., Schaffer, A.A., Zhang, J., Zhang, Z., Miller, W. and Lipman, D.J.: Gapped BLAST and PSI-BLAST: A new generation of protein database search programs, *Nucleic Acids Research*, Vol.25, No.17, pp.3389–3402 (1997).

[23] Altera DE2-115 Development and Education

Board, Terasic Technologies Inc., available from ⟨http://www.terasic.com⟩.
[24] 田中飛鳥，石川　淑，宮崎敏明：パイプライン化アレイプロセッサによる Smith-Waterman アルゴリズムの高速化，電子情報通信学会 VLSI 設計技術研究会，VLD2011-32, pp.141–146 (2011).
[25] Yamaguchi, Y., Maruyama, T. and Konagaya, A.: High Speed Homology Search with FPGAs, *Proc. Pacific Symposium on Biocomputing 2002*, pp.271–282 (2002).

推薦文

本研究では DNA 配列のシーケンスアライメントによく用いられる BLAST（Basic Local Alignment Search Tool）のハードウェアアルゴリズムを提案し，FPGA（Field-Programmable Gate Array）を用いて有効性を明らかにしている．従来のハードウェアを利用した高速化手法では，前処理を PC で行い，その後ハードウェアで主処理を行っていたが，PC とハードウェア間の通信によるボトルネックが発生し，ハードウェア化の効果は限定的であった．本研究では前処理部分を含めてすべてハードウェア化することにより前述のボトルネックを解消し，そのための効率的な情報フローとハードウェア回路設計を行った．提案手法は実験により検証され，ソフトウェアに比べて 800 倍以上の高速化が実証されている．このように本研究は DNA 配列のシーケンスアライメント処理の高速化に対して，大きな進歩をもたらすものであり，当該分野への大きな貢献が期待される．よって本論文を推薦する．

（東北支部長　小林広明）

石川 淑　（正会員）

2011 年会津大学コンピュータ理工学部卒業．2013 年同大学大学院修士課程修了．在学中，アレイプロセッサに関する研究に従事．

田中 飛鳥

2011 年会津大学コンピュータ理工学部卒業．2013 年同大学大学院修士課程修了．在学中，動的計画法に基づくアルゴリズムを汎用に扱えるアレイプロセッサに関する研究に従事．

宮崎 敏明　（正会員）

1981 年電気通信大学応用電子工学科卒業．1983 年同大学大学院修士課程修了．同年日本電信電話公社（現，NTT）入社．以来 LSI-CAD，通信用 FPGA および応用装置，アクティブネットワーク，ユビキタスネットワーク，センサネットワーク，アレイプロセッサの研究開発に従事．2005 年より会津大学コンピュータ理工学部教授．東京農工大学非常勤講師（2003～2007 年）．新潟大学大学院客員教授（2004 年）．電子情報通信学会先端オープン講座講師（1996～2008 年）．工学博士（東京工業大学）．IEEE シニア会員．電子情報通信学会会員．