Vol.2013-MPS-96 No.20
Vol.2013-BIO-36 No.20
2013/12/12

GPGPUを用いた高速な配列相同性検索の圧縮アミノ酸によるアルゴリズム改良

渡部 翔[1,a)]　鈴木 脩司[1,b)]　石田 貴士[1]　秋山 泰[1,2,c)]

概要：土壌や生体内などの環境に生息する遺伝子を同時に解析するメタゲノム解析では，大量で高感度な配列相同性検索を行わなければならず，膨大な計算時間を必要とする．従来用いられてきたBLAST等の配列相同性検索ツールは，現在のDNAシークエンサーのスループットに対して，計算速度は不十分である．そこで，我々は以前にGPUを用いた高速な配列相同性検索ツールを開発したがシークエンサーのスループットの更なる向上に伴い，一層の高速化が求められている．本研究では高効率且つ高感度に検索が可能になるとその有効性が示されている圧縮アミノ酸を利用した新たなアルゴリズムを提案し，GPU上で動作するより高速な配列相同性検索の実装を行った．また，その評価実験を行い，従来のツールと比較し，提案手法の有用性を証明した．

キーワード：相同性検索，GPGPU，圧縮アミノ酸

Improvement of a fast homology search tool using GPGPU by reduced amino acid alphabet

Watanabe Sho[1,a)]　Suzuki Shuji[1,b)]　Ishida Takashi[1]　Akiyama Yutaka[1,2,c)]

Abstract: Metagenomic analysis is a research to analyze genes obtained from an environment, and often requires large amount of sensitive homology searches which require immense amount of time. NCBI BLAST has been the most widely-used for this purpose, but its calculation speed is insufficient for a large amount of DNA sequence data obtained from current sequencers. In previous study, we developed a fast GPU-based homology search tool for metagenomic analysis. However, it is required further speed-up in accordance with improvement of the sequencers. Here, we propose a new and more efficient GPU-based homology search tool by using reduced amino acid alphabet. Additionally compared with several conventional tools, we establish the usefulness of the proposed tool.

Keywords: homology search, GPGPU, reduced amino acid alphabet

1. はじめに

単一生物のDNA配列を決定するゲノム解析に対して，海洋や土壌，腸内などに生息する多種の微生物のDNAを分離，培養を経ずに直接DNAを読み取り，環境中に生息する微生物の遺伝子の分布を解析する研究をメタゲノム解析という．メタゲノム解析では，環境中に含まれる微生物のすべてのゲノム配列が既知であることは稀であるため，遠縁のゲノム配列しか参照できないことが多い．そのため配列マッピングの際に多くの不一致やギャップを許容する必要がある．従来開発されてきたBWA[1], [2]やBowtie[3]等のマッピング手法では，検索の感度が不十分であり，メタゲノムの解析に利用することは困難である．

[1] 東京工業大学 大学院情報理工学研究科 計算工学専攻
Graduate School of Information Science and Engineering, Tokyo Institute of Technology
[2] 東京工業大学 情報生命博士教育院
Education Academy of Computational Life Sciences, Tokyo Institute of Technology
[a)] watanabe@bi.cs.titech.ac.jp
[b)] suzuki@bi.cs.titech.ac.jp
[c)] akiyama@.cs.titech.ac.jp

そういった理由から，メタゲノム解析では，一般に相同性検索と呼ばれる，より高感度な解析手法が利用されている．また，メタゲノム解析では検索の感度を向上させるために，DNA 配列のまま解析を行うのではなく，多くの場合，6 フレームのタンパク質配列に翻訳してから解析が行われる．DNA 配列は通常 A，T，G，C の 4 文字で表現される．これに対し，タンパク質配列は標準的な 20 文字で表現される．さらに，DNA 配列の比較は一致か不一致の 2 状態でしか区別しないことが多いが，タンパク質配列はアミノ酸間で性質の類似度に違いがあるため，アミノ酸ペア毎に置換スコアが付けられた置換行列が用いられる．このため，タンパク質配列のマッピングは DNA 配列のマッピングと比べ，計算はより複雑で困難なものとなっている．このようなタンパク質配列間での曖昧な一致も含めた配列の相同性検索では，もっとも正確なアラインメントが計算可能な Smith-Waterman アルゴリズム [4] が実装されている，SSEARCH[5] などを利用することが望ましいが，これは極めて低速であるため困難である．このため，現在では比較的高速な配列相同性検索が可能な近似的手法の BLAST[6], [7] がよく利用されている [8]．しかし，近年のスループットの向上した DNA シークエンサーは，BLAST の計算速度を超えて大量の DNA 断片配列を出力するため，すべての DNA 断片配列をタンパク質配列に翻訳してから配列相同性検索を行う解析では，計算時間が問題となっている．現在，最新の illumina 社の Hiseq2000（Hiseq2500）という DNA シークエンサーの出力は 1 ランで合計 600G 塩基にも達し，そのデータを解析するには BLAST を用いた場合，約 25000CPU 日が必要になると推定されている．

現在までに，BLAST 以外の高速な配列相同性検索のアルゴリズムは提案されており，BLAT[9] は良く知られているが，検索感度が低すぎるため実用的ではなかった．そこで，我々は以前に GPU を用いた高速な配列相同性検索ツール GHOSTM[10] を開発した．元来，GPU は画像処理に特化した計算ユニットであったが，現在は GPU の高い計算性能を描写処理だけに限らず汎用計算にも利用する動きが盛んになってきている．それらを GPGPU(General-Purpose computing on Graphics Processing Units) と呼び，近年では気象予測 [11] や流体シミュレーションなど様々な計算の高速化に貢献している．GHOSTM も相同性検索の高速化に GPU の恩恵を受け，BLAST の数十倍の高速化を達成した．一方，近年では GPU のようなアクセラレータの計算能力を利用するのではなく，アルゴリズムの改良による高速化手法も提案されている．suffix array と圧縮アミノ酸 [12] を用いて配列相同性検索を行うツール RAPsearch[13], [14] では，CPU のみで，高速且つ，高感度を達成している．しかし，シークエンサーのスループットの更なる向上に伴い，一層の高速化が求められている．

本研究は高効率且つ高感度に検索が可能になるとその有効性が示されている圧縮アミノ酸を利用した新たなアルゴリズムを提案し，GPU 上で動作する，より高速な配列相同性検索の実装を行った．またその評価実験を行い，従来のツールと比較し，提案手法の有用性を示した．

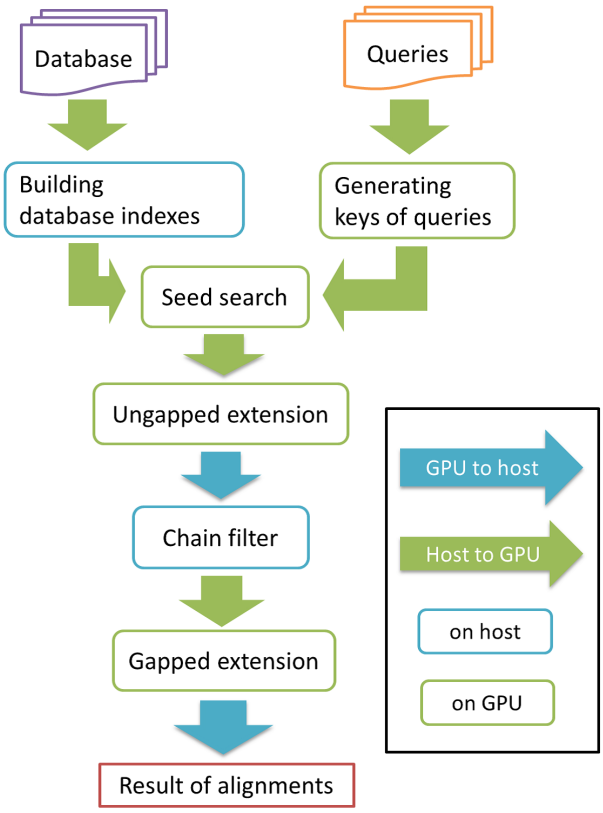

図 1　提案手法の処理の流れ
Fig. 1　Processing flow of proposed method.

2. 提案手法と実装

本研究で用いた圧縮アミノ酸はメモリ制約のある GPU 上で用いることにより，メモリ消費を抑え，検索の効率化に役立てることができる．節 2.2 でその詳細を述べる．GPU による実装は全て NVIDIA 社が提供する GPGPU のための統合開発環境 CUDA（Compute Unified Device Architecture）を用いて実装を行った．主要な計算の殆どを GPU 上で行うため，GPU 内でのメモリアクセス速度や並列化効率を向上させられるように実装を行った．

2.1 配列相同性検索の概要

本節では本研究で提案する手法の概要を説明する．基本的なアルゴリズムは BLAST と同様である．まず最初に，クエリとデータベース中の配列間において圧縮アミノ酸で表現した場合に完全一致している位置を探索する．その後，探索によって見つかった位置を中心にアラインメントの伸長（extension）を行う，seed-extension の手法を用いる．伸長の打ち切りには X-dropoff[7] を用いる．

提案手法の流れは以下の通りである．予め対象となる

Vol.2013-MPS-96 No.20
Vol.2013-BIO-36 No.20
2013/12/12

図 2　配列の連結
Fig. 2　Connecting sequences

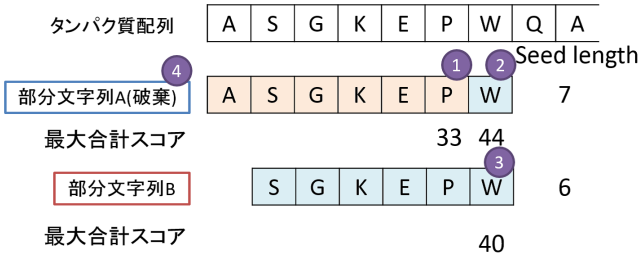

図 3　部分文字列の生成（$L_{min} = 6$，$L_{max} = 7$，$T_s = 39$）
Fig. 3　A example of generation of array substring ($L_{min} = 6$, $L_{max} = 7$, $T_s = 39$)

データベースから，圧縮アミノ酸に変換した部分文字列のインデックスを構築する．このとき，部分文字列の長さは圧縮アミノ酸内の最大スコアを用いて可変長となる．DNA シークエンサーによって読み取られた DNA 断片配列（queries）はタンパク質配列に翻訳し，データベースと同様に圧縮アミノ酸に変換した部分文字列キーを生成する．データベースと翻訳してできたタンパク質配列，二つの圧縮アミノ酸に変換した部分文字列で一致する位置（seed）を探索する．そして，探索して見つかった seed を中心にアラインメントの伸長を行い，スコアを計算し，相同性検索を行う．図 1 に提案手法の流れを示す．図 1 から分かる通り，主要な計算の殆どを GPU で行うことで，高速化を図った．

各処理の詳細を以降に述べる．

2.1.1　データベースの部分文字列インデックスの構築

相同性検索の対象であるタンパク質配列データベースには複数のタンパク質の配列が含まれている．しかし，GPU で処理を行う上で，複数の配列よりも単一の連続した配列の方が扱いやすいため，区切り文字 “#” を入れて，連結していく．図 2 にその様子を示す．このように単一の連結配列にすることで，データベース内の複数の配列に対して，まとめて seed 探索を行うことができる．

次に連結した配列を 1 文字ずらしで圧縮アミノ酸に変換した部分文字列を生成する．そして，その部分文字列の出現位置のインデックスを構築していく．部分文字列の生成には 3 つのパラメータ（1. 最小長 L_{min}，2. 最大長 L_{max}，3. 長さ決定のためのスコア閾値 T_s）を用いて行う．図 3 は $L_{min} = 6$，$L_{max} = 7$，$T_s = 39$ のときの部分文字列生成例である．図内の番号 1，2，3，4 は以降の説明順（インデックスの構築の流れ）である．1. まず最初に，部分文字列 A の最小長 6 文字を取り出す．このとき，圧縮アミノ酸で完全一致したときのスコアは図 6 のグループ内完全一致文字最大スコアを用いて，33 と求められる．2. 長さ 6 の部分文字列 A はスコア閾値にはまだ届かないので，もう一文字追加する．このとき，長さ 7 でスコア閾値を超えたので，これを部分文字列とする．またこのパラメータの場合，最大長 7 という条件も満たしているので，スコアに関係なく，部分文字列となる．3. 次の部分文字列 B では最小長 6 を取り出したとき，スコアは 40 と求められる．このとき部分文字列 B のスコアはスコア閾値を超えているので，これ以上文字を追加することなく，長さ 6 の部分文字列とする．4. ここで，部分文字列 B は部分文字列 A に包含されるため，この位置では部分文字列 B の方が，より多くのアラインメント候補が見込めることが分かる．従って，部分文字列 A は無駄な部分文字列となるので破棄する．

以上のように部分文字列のインデックスを構築することで，より効率的な検索を行うことができる．

2.1.2　DNA 断片配列の翻訳と部分文字列キーの生成

DNA シークエンサーによって出力された DNA 断片配列を 6 フレームに翻訳し，翻訳して出来た 6 つのタンパク質配列をデータベース同様に連結する．この連結した配列を 1 つのクエリとする．また DNA シークエンサーによって出力された DNA 断片配列も複数あるので，GPU で処理を行う上では，データベース同様単一の連続した配列の方が扱いやすいため，翻訳して出来た 6 つのタンパク質配列を連結した配列をさらに連結する．よって大量のクエリを同時にまとめて seed 探索出来る．

クエリの部分文字列キーはデータベースと同様に 3 つのパラメータを用いて生成される．さらにデータベースは 1 文字ずらしで部分文字列を生成していたのを，1 文字に限らず s 文字ずらしで生成することができる．

2.1.3　Seed 探索と Ungapped Extension

seed 探索では，データベースと翻訳してできたタンパク質配列，二つの圧縮アミノ酸に変換した部分文字列キーで完全一致する位置を探索する．一つでもマッチがあった位置はアラインメントの候補位置として記録し，次の伸長（extension）に移る．伸張ではまず最初に，seed 探索でアラインメントの候補となった位置を中心にして ungapped extension を行う．ungapped extension は BLAST と同様にスコアが低下し始めたら，伸長を停止する X-dropoff を使用している．このときのスコアが T_g を超えた候補のみ，次の gapped extension の候補として記録しておく．

GPU にこれらを実装するとき，seed 探索では予めどれだけの候補数が得られるか予測できない．そのため，一度に seed 探索と ungapped extension を行おうとすると，GPU の global memory の大きさを超えてしまう可能性がある．従って本研究では，2 段階に分けてこれらの操作を行った．

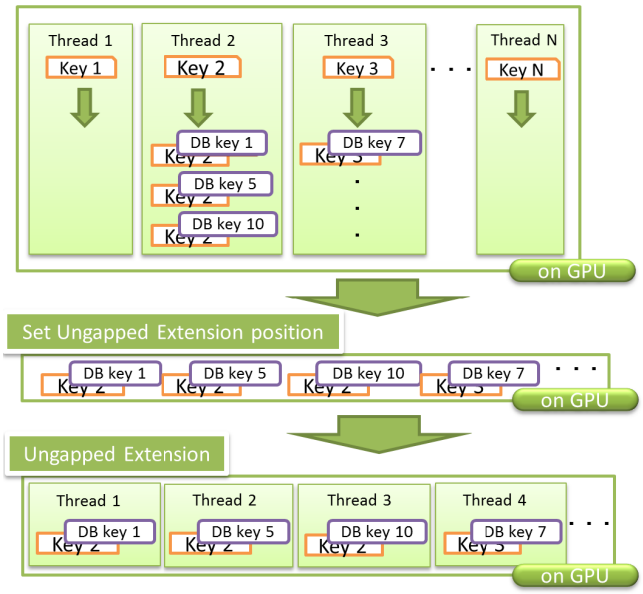

図 4　GPU 上の ungapped extension の流れ
Fig. 4　Processing flow of ungapped extension on GPU

(1) クエリの部分文字列キー毎のアラインメント候補数をカウント
(2) global memory を超えない数ずつ ungapped extension を行い，全てのアラインメント候補を計算するまで繰り返す

このようにすることによって GPU 上でもメモリが不足することなく，seed 探索と ungapped extension が行うことができる．

また ungapped extension の並列化効率を向上させるために，図 4 に示すような前処理（Set Ungapped Extension Position）を追加した．追加しない場合，クエリの部分文字列キー毎の thread によって計算の重みが異なってしまう．それを各候補毎に 1 thread が割り当てられるようにすることで，この問題を解決した．

2.1.4　Gapped Extension

gapped extension も ungapped extension と同様に X-dropoff を使用して，スコア計算を停止する．その後，クエリ毎にスコアの高いものをまとめ，相同性検索の結果として出力する．

gapped extension は Smith-Waterman アルゴリズムをベースにして計算される．Smith-Waterman アルゴリズムの GPU 化は既に CUDASW++ 2.0[16] 等が提案され，GPU によって高速に BLAST 以上の速度で計算できることが知られている．これらはある程度長い配列を想定しており，thread を同期させながら計算する手法をとっている．しかし今回我々が想定している配列は 150 塩基程度の短い配列であるため，thread を同期させながら計算すると，同期に時間がかかってしまうため, 伸張方向毎に thread を割り当てた．

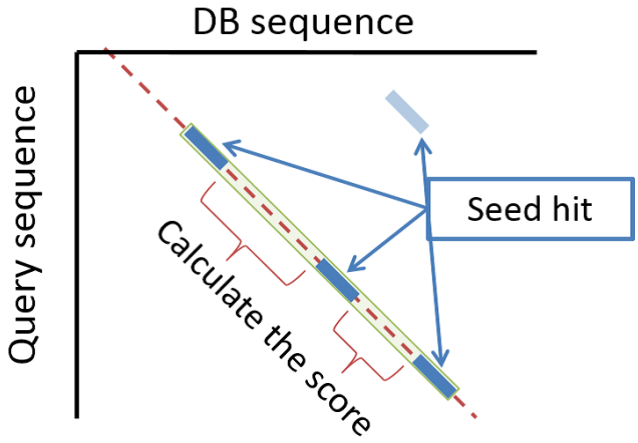

図 5　チェインフィルター
Fig. 5　Chain Filter

2.1.5　Chain Filter

ungapped extension 終了後，データベース配列とクエリ配列の対角線に gapped extension の候補となる seed が並ぶことがある．これらの間を計算し，スコアが X-dropoff パラメータよりも低下せずに繋がるのであれば，これらは gapped extension において無駄なアラインメント候補となる．よってこれらの seed をまとめ，一つの長い seed として扱い，gapped extension の計算量を減らすことができる．図 5 はチェインフィルターの例である．

チェインフィルターは，アラインメント候補の整理に伴うメモリサイズの変更処理などのために GPU での実行に適しておらず，CPU で実行する．

2.2　圧縮アミノ酸

本節では本研究で使用した圧縮アミノ酸の詳細について説明する．圧縮アミノ酸は様々なタンパク質の研究のために考案されてきたものである．圧縮アミノ酸には，Murphy LR, *et al.*（2000）[12] によって考案されたものを用いた．この圧縮アミノ酸は 20 種類の各アミノ酸同士の類似度が記述された置換行列 BLOSUM より導出されたものである．使用した圧縮アミノ酸は図 6 のようにグルーピングされる．図 6 のグループ内完全一致文字最大スコアとは，圧縮された各アミノ酸グループ内の文字で完全一致のときに最大スコアを取るもののスコアである．圧縮アミノ酸を用いる効果として，曖昧な一致を許容し感度の向上が期待出来る．部分文字列の探索に用いることにより，長い部分文字列で感度を落とさずに利用可能となり，検索全体の計算効率を向上させることが期待出来る．

我々は 10 グループの圧縮アミノ酸パターンを用いた．10 グループのパターンを用いた理由は二つある． 1 つ目に文献 [13] の中で，圧縮アミノ酸パターンの性能評価が行われ，そこで感度の低下を最小限に抑え，高効率で検索ができることが証明されているからである．二つ目に GPU に

Vol.2013-MPS-96 No.20
Vol.2013-BIO-36 No.20
2013/12/12

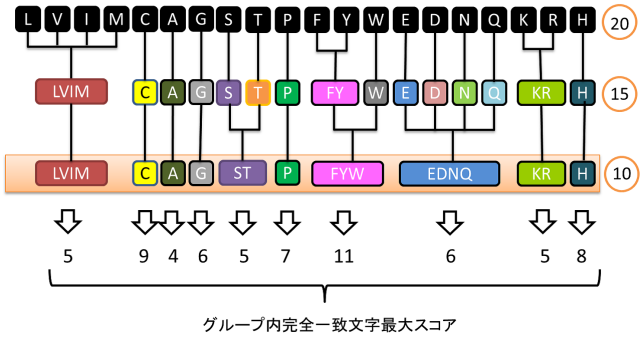

図 6　圧縮アミノ酸と各グループ内の
完全一致文字最大スコア（BLOSUM62）

Fig. 6　Reduced amino acid alphabet and maxmum score of exact match character in the group（BLOSUM62）.

実装する上で，global memory のサイズに制約があるためである．部分文字列のインデックスを作成する際に，標準的な 20 文字の場合，$(2^5)^{(length\ of\ substring)} \times 4$ byte のメモリを使う．部分文字列の長さが 7 であれば 128GB ものメモリを使うこととなる．これはメモリサイズが高々 6GB の GPU を大きく超える．一方 10 文字に圧縮した場合は $(2^4)^{(length\ of\ substring)} \times 4$ byte のメモリで済み，部分文字列の長さが 7 ならば 1GB のメモリで良い．

2.3　GPU 内でのメモリアクセスの高速化

各処理でメモリアクセス頻度の高い，各アミノ酸毎の置換スコアが記述されている置換行列を texture memory に配置した．このようにすることで，R0 cache の効果により global memory に配置するよりも高速にアクセスすることが可能となる．またクエリの部分文字列キーを生成する処理において，圧縮アミノ酸への文字変換テーブルも全ての thread が何度もアクセスするため，圧縮アミノ酸への文字変換テーブルも texture memory に配置することとした．

3.　性能評価実験

本研究で提案した手法の相同性検索の計算速度と感度を評価する実験を行った．実験に使用した計算機は TSUBAME 2.5 [15] の Thin ノードである．このノードの CPU は Intel Xeon processor X5670（6 core, 2.93GHz）を 2 つ，GPU は NVIDIA Tesla K20X を 3 つ，メモリは 54GB である．OS は SUSE Linux Enterprise Server 11 SP1，gcc は version 4.3.3，GPU を使用するために CUDA 5.0 を使用した．

性能比較には NCBI BLAST version 2.2.26，BLAT version 34，RAPsearch version 2.12，GHOSTM version 1.2.1 を使用した．計算に用いた置換行列はすべて BLOSUM62 を用いた．BLAST，RAPsearch，提案手法ではクエリの複雑度の低い領域をフィルタするオプションを使用した．GHOSTM はオプションでそのフィルタを使用できないため計算時間はその他のツールよりもかかっていると推定される．BLAST は open gap penalty を 11，extend gap penalty を 1 とした．また TSUBAME 2.5 の実行限界時間の都合で，12 threads で実行を行った．具体的に使用した BLAST のオプションは

“`-p blastx -m 8 -b 1 -v 1 -G 11 -E 1 -g T -f T -M BLOSUM62 -a 12`”

である．BLAT は予め DNA 断片配列をタンパク質配列に翻訳しておき，それをクエリとして利用した．使用した BLAT のオプションは

“`-q=prot -t=prot -out=blast8`”

である．RAPsearch は初期値のオプションに 6 threads（1 CPU socket）で実行を行った．オプションは “`-z 6`” である．GHOSTM のオプションも初期値（“`-r 8 -e 2 -G 11 -E 1`”）を用いて実行した．提案手法の部分文字列の生成パラメータは seed 最小長 $L_{min} = 6$，最大長 $L_{max} = 7$，長さ決定のためのスコア閾値 $T_s = 33, 36, 39, 42$ を使用した．クエリの部分文字列キーを生成する際，s 文字ずらしで区切っていくかのパラメータは $s = 1, 2$ を使用した．そのほかのパラメータの値は BLAST の値と共通のものを用いた．

最初に提案手法のパラメータ T_s, s を変化させ，それぞれの計算速度と感度の評価を行った．次に提案手法のパラメータを変化させ実験した内，感度が RAPsearch とほぼ同じになり，その中で最も高速なパラメータを選択してその他のツールと比較を行った．

3.1　使用データ

タンパク質配列データベースは 2013 年 5 月時点の KEGG Genes（genes.pep）[17], [18], [19] を使用した．このデータベースはタンパク質のタンパク質配列を約 1024 万本を含み，ファイルサイズは約 3.9GB となる．検索に用いたクエリデータは，NCBI Sequence Read Archive から取得した SRR407548 を使用した．SRR407548 は最新のシークエンサーである illumina 社の Hiseq2000 で読み取った土壌メタゲノムである．1 本あたりの長さは 150 塩基であり，順鎖だけでファイルサイズは 38GB ある．全データを利用した場合，計算に多くの時間が必要になるため，本研究では SRR407548 の順鎖からランダムで 10 万本を選択して使用した．

3.2　提案手法のパラメータ T_s, s を変化させたときの性能評価

本節では提案手法のパラメータ T_s, s を変化させたときの計算時間と感度の評価を行った．感度の評価には，DNA 断片配列 1 本毎の検索の正確さを評価した．そのために，最適アラインメントを計算する SSEARCH の結果を使用し，DNA 断片配列毎に SSEARCH の結果と同じデータベース

表 1　提案手法のパラメータによる相同性検索の計算時間（sec.）の変化

Table 1　Computation time （sec.） on various parameter of proposed method

パラメータ	計算時間（sec.）
$T_s = 33, s = 1$	5,053.45
$T_s = 36, s = 1$	3,371.42
$T_s = 39, s = 1$	2,317.91
$T_s = 42, s = 1$	2,001.85
$T_s = 33, s = 2$	2,753.10
$T_s = 36, s = 2$	1,913.01
$T_s = 39, s = 2$	1,347.27
$T_s = 42, s = 2$	1,173.70

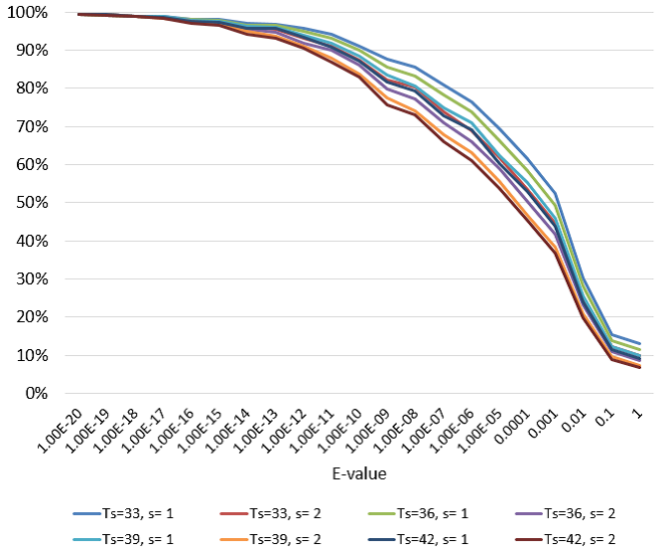

図 7　提案手法のパタメータによる E-value を変化させた場合の正解が含まれる割合（%）の変化

Fig. 7　Correct alignment rate for each E-value （%） on various parameter of proposed method

内のタンパク質配列が最もスコアが高い場合を正解とした．実験には 1 thread + 1 GPU 用いた．

計算時間を表 1，感度を図 7 に示す．E-value は E^{-21} 以下は省略した． T_s を固定して s を 1 から 2 にすると，計算速度は 1.7 から 1.8 倍程度に高速化することが分かった．しかし s を増加させると高速化される代わりに，seed 探索が粗くなるため，感度は低下した．感度の低下の差は E-value の E^{-14} 以上からパラメータによって開くことが分かる．感度は T_s が低いほど高いことが分かるが，これは T_s が低いほど短い部分文字列キーの数が増加するためである．しかし感度が向上する一方，アラインメントの候補の数が増え計算速度は低下してしまう．

3.3　既存手法との計算速度比較

相同性検索の計算時間を評価するために，計算時間と BLAST（12 threads）を 1 としたときの計算速度比を用いて比較を行った．提案手法は RAPsearch とほぼ同じ感度になり，その中で最も高速なパラメータ $T_s = 36, s = 2$ を選択して比較を行った．

表 2 に示すように提案手法は BLAST 12 threads と比較して，61.5 倍の高速化を達成した．また今回 BLAST は 12 threads でしか実行できなかったが，1 thread を 12 threads の実行時間から逆算して 12 倍の実行時間だったとすると，提案手法は BLAST 1 thread と比較して，738.3 倍の高速化されることが推定される．その他のツールと比較しても，BLAT より 14.3 倍，RAPsearch より 1.5 倍，GHOSTM より 5.6 倍高速化を達成した．

表 2　相同性検索の計算時間（sec.）と BLAST との計算速度比

Table 2　Computation time（sec.）and accelaration ratio with respect to the BLAST

ツール	計算時間（sec.）	BLAST（12 threads）からの速度比
提案手法（1 thread + 1 GPU）	1,913.01	61.52
BLAST（12 threads）	117,692.13	1.00
BLAT（1 thread）	27,378.19	4.30
RAPsearch（6 threads）	2,818.25	41.76
GHOSTM（1 thread + 1 GPU）	10,784.69	10.91

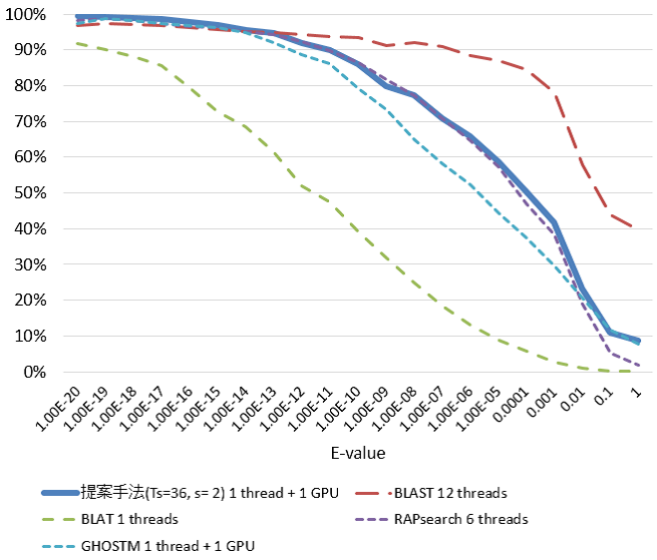

図 8　E-value を変化させた場合の正解が含まれる割合（%）

Fig. 8　Correct alignment rate for each E-value（%）.

3.4　既存手法との感度比較

評価方法は節 3.2 と同様の方法を用いた．提案手法は 3.3 と同じパラメータ $T_s = 36, s = 2$ を選択し比較を行った．結果を図 8 に示す．E-value の E^{-21} 以下は省略した．

図 8 より，提案手法は E-value が E^{-11} までは BLAST とほぼ同じ感度が出ていることが分かった．E^{-11} よりも高い E-value で感度に差がでたのは提案手法，RAPsearch ともに BLAST よりも長い部分文字列（$L_{min} = 6$ ～）の

検索を行っているためである．また提案手法は BLAT や GHOSTM よりも常に高い感度を維持していることが分かる．これは圧縮アミノ酸によって曖昧な配列を許容した seed 探索を行っているためである．RAPsearch と比較しても，同等以上の感度を達成している．これは seed 探索の際に，RAPsearch は提案手法よりも長い seed を許容しているため，その際に類似性の高い配列を見逃していると考えられる．

4. 結論

4.1 本研究の成果

本研究では，GPU を用いた配列相同性検索ツールに圧縮アミノ酸による新たなアルゴリズムを提案し，その実装と評価実験を行った．RAPsearch とほぼ同感度のパラメータを用いて，BLAST 12 threads と比較して，61.5 倍の高速化を達成した．また 1 thread と比較すると 738.3 倍の高速化されることが推定される．実験した最速パラメータと BLAST を比較すると，12 threads で 100.3 倍，1 thread で 1203.3 倍の高速化を達成した．高速・高感度な配列相同性検索ツールである RAPsearch 6 threads（1 CPU socket）と比較して同等以上の感度で 1.2 倍から 1.5 倍の高速化を達成した．以前に我々が開発した GPU を用いた配列相同性検索ツール GHOSTM と比較して，常に高い感度を維持し，最大で 9.2 倍の高速化を達成した．

4.2 今後の課題

シークエンサーのスループットの更なる向上のため，一層の高速化が望ましいが，現状の課題として，seed の候補が多く，ungapped extension が律速となっている．したがって高速化のためには，この ungapped extension の更なる改良が必要である．

また今後は，読み取られる DNA 断片配列の長さも長くなっていくと考えられる．現在の GPU の gapped extension は条件分岐が多く GPU には向かない．従って，断片配列が長くなればなるほど，gapped extension の計算量は増え，計算に時間がかかると予想されるため新たな計算手法が求められる．

謝辞 本研究は NVIDIA CUDA Centers of Excellence (CCOE)，文部科学省 博士課程教育リーディングプログラム「情報生命博士教育院」の支援を受けて行われたものである．

参考文献

[1] Li H, Durbin R: “Fast and accurate short read alignment with Burrows-Wheeler transform”, *Bioinformatics*, 25(14):1754–1760(2009).
[2] Li H, Durbin R: “Fast and accurate long-read alignment with Burrows-Wheeler transform”, *Bioinformatics*, 26(5):589–595(2010).
[3] Langmead B, Trapnell C, Pop M, Salzberg SL: “Ultrafast and memory-efficient alignment of short DNA sequences to the human genome”,*Genome Biology*, 10(3):R25(2009).
[4] Smith TF, Waterman MS: “Identification of Common Molecular Subsequence”, *Journal of Molecular Biology*, 147(1):195–197(1981).
[5] Pearson WR: “Searching protein sequence libraries: comparison of the sensitivity and selectivity of the Smith-Waterman and FASTA algorithms”, *Genomics*, 3:635–650(1991).
[6] Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ: “Basic local alignment search tool”, *Journal of Molecular Biology*, 215:403–410(1990).
[7] Altschul SF, Madden TL, Schaffer AA, *et al*: “Gapped BLAST and PSI-BLAST: a new generation of protein database search programs”, *Nucleic Acids Research*, 25(17):3389–3402(1988).
[8] Dalevi D, Ivanova NN, Mavromatis K, *et al*: “Annotation of metagenome short reads using proxygenes”, *Bioinformatics*, 24(16):7–13(2008).
[9] Kent WJ: “BLAT–the BLAST-like alignment tool”, *Genome Research*, 12(4):656–664(2002).
[10] Suzuki S, Ishida T, Kurokawa K, Akiyama Y: “GHOSTM: A GPU-Accelerated Homology Search Tool for Metagenomics”, *PLoS ONE*, 7(5):e36060(2012).
[11] Shimokawabe T, Aoki T, Muroi C, *et al.*: “An 80-Fold Speedup, 15.0 TFlops Full GPU Acceleration of Non-Hydrostatic Weather Model ASUCA Production Code”, *SC10 Proceedings of the 22nd Annual International Conference for High Performance Computing Networking Storage and Analysis. IEEE Computer Society*, 1–11(2010).
[12] Murphy LR, Wallqvist A, Levy RM: “Simplified amino acid alphabets for protein fold recognition and implications for folding”,*Protein Eng*, 13(3):149–152(2000).
[13] Ye Y, Choi JH, Tang H: “RAPSearch: a fast protein similarity search tool for short reads”, *BMC Bioinformatics*, 12:159(2011).
[14] Zhao Y, Tang H, Ye Y: “RAPSearch2: a fast and memory-efficient protein similarity search tool for next-generation sequencing data”, *Bioinformatics*, 23(1):125–126(2012).
[15] TSUBAME 2.5 — [GSIC] 東京工業大学学術国際情報センター. http://www.gsic.titech.ac.jp/tsubame2
[16] Liu Y, Schmidt B, Maskell DL: “CUDASW++2.0: enhanced Smith-Waterman protein database search on CUDA-enabled GPUs based on SIMT and virtualized SIMD abstractions”,*BMC Research Notes*, 3:93(2010).
[17] Kanehisa M, Goto S, Furumichi M, Tanabe M, Hirakawa M: “KEGG for representation and analysis of molecular networks involving diseases and drugs”, *Nucleic Acids Research*, 38:355–360(2010).
[18] Kanehisa M, Goto S, Hattori M, Aoki-Kinoshita KF, Itoh M, Kawashima S, Katayama T, Araki M, Hirakawa M: “From genomics to chemical genomics: new developments in KEGG”, *Nucleic Acids Research*, 34:354–357(2006).
[19] Kanehisa M, Goto S: “KEGG: Kyoto Encyclopedia of Genes and Genomes”, *Nucleic Acids Research*, 28:27–30(2000).