Vol.2011-BIO-25 No.32
2011/6/24

GPUを用いた高速な配列相同性検索のsuffix arrayによる改良

鈴木脩司[†1]　石田貴士[†1]　秋山泰[†1]

我々は以前にGPUを用いたメタゲノムの配列マッピングの手法を開発し，GHOSTMというツールとして実装を行った．しかし，短いDNA断片配列に対して設計されたGHOSTMでは長いDNA断片配列を解析するには不向きである．また，1ノード当たりのCPUのコア数とGPUの枚数がほぼ等しいと仮定していたが，多くの計算機ではノード当たりのGPUの枚数より多くのCPUコアが搭載されており，残されたCPUコアが活用出来ていなかった．これらの課題を解決するために，以前開発したsuffix arrayを用いたseedの探索を使った相同性検索の手法をベースとし，見つかったseedに対してGPUを用いたSmith-Waterman法ベースのアライメントの伸長をCPUによるseed探索と非同期に行うことで長い配列にも対応可能で1ノードの全ての計算資源を活用するツールを開発した．

Improvement of a fast homology search tool on GPUs with suffix array

Shuji Suzuki,[†1] Takashi Ishida[†1]
and Yutaka Akiyama[†1]

We had developed a homology search tool for metagenomics as *GHOSTM*. However, the tool is designed for a search for short query sequences, and unsuitable to search for long query sequences. Also, the tool uses the same number of CPU cores and GPUs. However a node has more CPU cores than GPUs on most of computer system. Therefore, GHOSTM does not utilize the all resources of a node. To solve these problems, we improved the previously developed system, which uses suffix array for the seed search, and accelerated the seed extension part based on Smith Waterman algorithm by using GPUs calculation. As results, the tool can use all calculation resources of a computation node and efficiently works even for long query sequences.

1. はじめに

近年，次世代DNAシーケンサーの登場により大量のDNAデータが得られるようになった．次世代DNAシーケンサーは1ランで数千億塩基以上が解読可能であり，これは以前のDNAシーケンサーと比べると数万倍のスループットである．しかし，この次世代DNAシーケンサーによって得られるデータはDNAの断片配列であるため，その情報を利用するためには配列マッピングなどの解析処理が必要となる．配列マッピングとはDNAシーケンサーによって読み取ったDNA断片配列がリファレンスゲノムのどの部分のものであったかを同定する処理であり，配列マッピングを行うツールとしてはBWA[1),2)]やBowtie[3)]などが開発れてきた．これらの配列マッピング手法は高速であるが，DNA断片配列とリファレンスゲノムとの間で数塩基程度の違いしかないような高い一致の場合にのみマッピングが可能であり，遠縁の相同配列間のように，多くのギャップや置換が含まれるような曖昧な一致がある場合には検索は不向きである．

一方，土壌や腸内等に生息する複数の微生物の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メタゲノム解析という研究がある．この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，メタゲノムの配列マッピングを行うには同じ種のゲノム情報を参照する必要があり，リファレンスゲノムのある単一の生物に対する通常の配列マッピングを行うのに比べて，配列マッピングの際に多くのミスマッチやギャップを許容する必要がある．そのため，従来開発されてきた高速な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のは困難である．そういった理由から，メタゲノム解析では一般に相同性解析と呼ばれるより高感度な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増すためにDNA配列のまま行うのではなく，多くの場合アミノ酸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文字で表現されているのに対してアミノ酸配列は標準的な20文字で表現されている．さらに，DNA配列での比較はマッチかミスマッチの2状態でしか区別しないことが多いが，アミノ酸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，アミノ酸配列で配列マッピングを行うのはDNA配列の場合の配列マッピン

†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

グと比べて計算はより複雑なものとなっており，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メントが計算可能な Smith-Waterman アルゴリズム[4] の実装である SSEARCH[5] 等を利用することが望ましいが，これは極めて低速であるという問題がある．このため，現在では比較的高速な配列比較が可能な近似的手法の BLAST[6),7)] が利用されている．[9)] しかし，次世代 DNA シーケンサーは大量の DNA 断片配列を出力するため，すべての DNA 断片配列をアミノ酸配列に翻訳してから配列マッピングを行うのは BLAST を用いても長い計算時間が必要となる．現在，最新の illumina 社の Hiseq 2000 という DNA シーケンサーの出力は 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約 25,000CPU 日が必要になると推定される．BLAST 以外にも高速な相同性配列のアルゴリズムは提案されており，より高速な配列比較を行うツールとしては BLAT[8)] が良く知られている．しかし，BLAT の検索は高速であるが配列検索の感度が低すぎるため残念ながらメタゲノム解析では実用的とは言い難い．そのため，メタゲノム解析のための高感度で高速な相同性検索手法が必要とされている．このため，我々は以前，GPU を用いて相同性検索を行うことで高速にメタゲノムの配列マッピングを行うツールである GHOSTM を開発した．[10)] 元来，GPU は画像処理に特化したチップであったが，GPU を描画処理だけでなく汎用計算にも利用すされるようになっており，GPGPU（General-Purpose computing on Graphics Processing Units）と呼ばれ，近年では気象予測など様々な計算の高速化に利用されている．[11)] GPU 計算によって処理を高速化することで GHOSTM では BLAST に近い感度での相同性検索が約 40 倍高速に実行可能となり，これによって高速なメタゲノムの配列マッピングが可能となった．

しかし，まだいくつか問題点がある．DNA シーケンサーの出力する DNA 断片配列長は徐々に伸びている傾向にあり，短い DNA 断片配列に対して設計された GHOSTM では長い DNA 断片配列を解析するには不向きである．また，GHOSTM は 1 ノード当たりの CPU のコア数と GPU の枚数がほぼ等しいと仮定して，1 CPU コア対 1 GPU の割合を使用して計算を行う．しかし，CPU のコアは年々増加傾向にあり，多くの計算機では GPU の枚数と CPU のコア数には大きな隔たりがある．昨年，TOP500 で世界 4 位となった東京工業大学のスーパーコンピュータである TSUBAME2 では 1 ノードあたり 12 コア，3GPU という構成になっている．[12)] このため，1 CPU コア対 1GPU の割合で使用している GHOSTM では 1 ノードの性能を十分に引き出せない．

このため，本研究ではこれらの問題を解決するために以前に開発した suffix array[13)] を用いた候補探索の手法[14)] と組み合わせ，長い DNA 断片配列が解析でき，マルチ CPU コア，1GPU という組み合わせで使用することを想定した手法を提案し，実装を行った．

2. 手　　法

まず，本研究で提案する手法の概要を説明する．基本的には BLAST と同様にクエリとデータベース中の配列間で局所的にスコアの高くなる位置を探索（seed 探索）をした後，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．

提案手法の流れは以下の通りである．

予め対象となるデータベースの suffix array を構築しておく．DNA シーケンサーによって読み取られた DNA 断片配列はアミノ酸配列に翻訳し，データベースと同様に suffix array を構築する．データベースと翻訳してできたアミノ酸配列，2 つの suffix array を用いて，局所的にスコアの高くなる位置 (seed) を探索する．そして，探索して見つかった seed を中心にアライメントの伸長を行い，スコアを計算する．

DNA 断片配列の長さが長くなるにつれて伸長の計算に多くの時間が掛かるようになるため，GPU は伸長を行いアライメントのスコアを計算する部分に使用した．伸長の部分は基本的には Smith-Waterman アルゴリズムが使われている．Smith-Waterman アルゴリズムの GPU 化は既に CUDASW++[15),16)] 等が提案され，GPU によって高速に計算できることが知られている．Seed 探索の計算は多くのメモリが必要となり，メモリの少ない GPU には不向きであるため，本研究では CPU で計算を行っている．このため，我々は seed 探索はマルチ CPU で計算し，伸長は 1 つの GPU が計算するようにした．

2.1 Suffix array

本研究で使用した suffix array の詳細について説明する．Suffix array は全文検索などに利用される検索アルゴリズムの一つであり，ある文字列 T が与えられたとき T のすべての接尾辞に添字を付け辞書順に並べ替えることによって得ることができる．**図 1** に例を示す．ここで，Σ を文字の集合とし T は Σ の要素による配列であるとする．T の末尾には配列の終端を表す $\$ \notin \Sigma$ かつ $\$ < \forall c \in \Sigma$ となる\$を付けている．この suffix array の構築には $O(|T|)$ で実行できるアルゴリズムが提案されている．また，文字列 P の完全一致の検索は最悪 $O(|P|log|T|)$ で実行できることが知られている．

2.2 データベースの suffix array の構築

配列マッピング先のアミノ酸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．このアミノ酸配列毎に個々の suffix array を構築するのではなくアミノ酸配列を連結さ

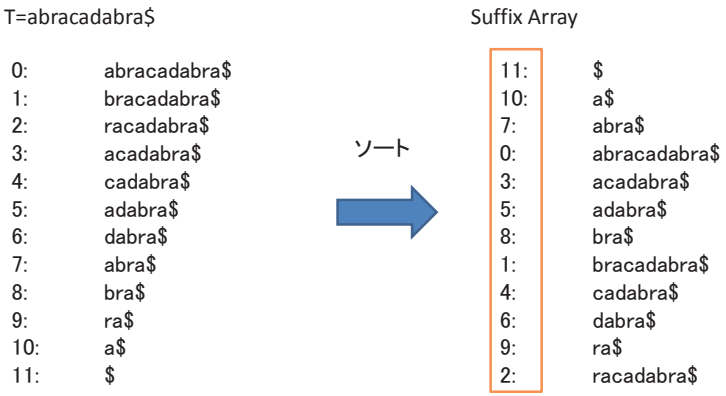

図 1 Suffix Array
Fig. 1 An example of suffix array

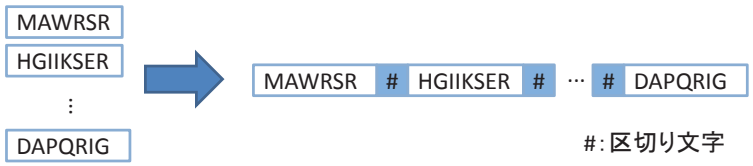

図 2 配列の連結
Fig. 2 Connecting sequences

せ，単一の配列にした後，連結配列に対して suffix array を構築する．このように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．連結をする際には**図 2** のようにアミノ酸配列中には存在しない文字 # を区切り文字として，配列と配列の間にこの区切り文字を挿入して連結していく．

2.3 DNA 断片配列の翻訳と suffix array の構築

DNA シーケンサーによって出力された DNA 断片配列は 6 フレームで翻訳し，翻訳して出来た 6 つのアミノ酸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今後，一つのクエリとする．データベースの時と同様に，配列を連結することにより，翻訳して出来た 6 つのアミノ酸配列をまとめて seed 探索ができる．

2.4 クエリとデータベースの suffix array を用いたアラインメントの seed 探索

クエリとデータベースの suffix array を用いてスコアが閾値 T_s 以上になるクエリとデータベースの部分文字列のペアを探索し，部分文字列の位置を seed とする．**図 3** にクエリとデータベースの suffix array を用いたアラインメントの seed 探索をする擬似コードを示す．ここで L_{max} は探索する文字列の最大長，$SASearch(SA, P)$ はある文字列 T の suffix array(SA_T) を用いてソートされた T の接尾辞中で接尾辞の先頭に文字列 P が出現する suffix array 上での範囲 sp, ep を返す関数，$maxScore$ は w_q を元にして w_d が w_q と完全に一致した場合に得られるスコア，D は $maxScore$ と $score$ の差をどれだけ許すかを示す値，$S[c, c']$ は文字 c, c' に対する置換行列の値である．D が 1 のとき，完全一致のクエリとデータベースの部分文字列のペアのみを検索するようになる．

BLAST では短い固定長の部分文字列とその部分文字列から数文字置換した近傍の部分文字列を用いて探索を行うが，固定長では感度を良くするために近傍の部分文字列を列挙する際の閾値を低くしなければならない．そうすると，探索する近傍の文字列の量が増えてしまい探索時間が増加する．しかし，本研究で用いた探索では suffix array を用いることで任意の長さの検索を行える特徴を生かし，閾値 T_s 以上の任意の長さの部分文字列を検索する．こうすることで，一致率の高い文字列では短い文字列が一致した時点でヒットとし，また一致率の低いものは長い文字列が一致した時点でヒットとなることでヒットする量を減らしつつ探索の感度を保つことが可能となっている．また，探索時間を短縮するためにクエリとデータベースの両方で suffix array を用いることで共通の部分文字列に関してはまとめて探索を行う．

2.5 ギャップサイズを固定した伸長

BLAST ではスコアが低下し始めると伸長を終了する X dropoff[7] が使用されている．このため，伸長を計算する際に必要となるメモリの量が一定ではない．GPU のメモリは CPU メモリに比べて少なく，GPU で伸長の部分を計算する際，GPU の性能を引き出すために，多くのスレッドを同時に使用する．この時，すべてのスレッドが伸長を計算するのに十分なメモリを確保しようとするとメモリが不足し，同時に実行できるスレッド数が減少する．このため，我々は**図 4** のように伸長を開始する位置の対角線から最大で w 個までギャップを許容するようにした．こうすることで，クエリの長さが長くなった場合でも，使用するメモリ量を $2w + 1$ と一定になり，限られた GPU のメモリで計算が可能となる．

```
SeedSearch(w_q, w_d, maxScore, sscore)
 1: if |w_q| < L_max then
 2:   for all c ∈ Σ do
 3:     w'_q ⇐ w_q + c
 4:     sp, ep ⇐ SASearch(SA_q, w'_q)
 5:     if sp ≤ ep then
 6:       maxScore' ⇐ maxScore + S[c, c]
 7:       for all c' ∈ Σ do
 8:         w'_d ⇐ w_d + c'
 9:         sp', ep' ⇐ SASearch(SA_d, w'_d)
10:         if sp' ≤ ep' then
11:           score' ⇐ score + S[c, c']
12:           if score' ≤ T_s then
13:             sp, ep, sp'ep' を保存する
14:             return
15:           else if score' > maxScore' − D and score' > 0 then
16:             Searh(w'_q, w'_d, maxScore', score')
17:           end if
18:         end if
19:       end for
20:     end if
21:   end for
22: end if
```

図 3 クエリとデータベースの suffix array を用いたアラインメントの seed 探索
Fig. 3 Search for candidate positions of optimum alignment using suffix arrays for queries and database

2.6 GPU の非同期実行

CPU と GPU は非同期で通信，計算が可能である．このため，seed 探索をした後，GPU で伸長を計算させている最中に CPU が GPU の計算を終了するのを待っている必要はなく，

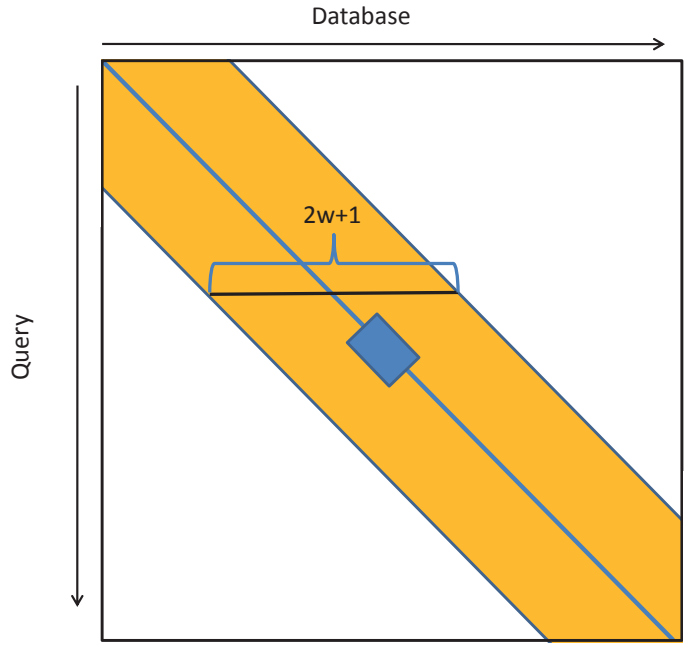

図 4 伸長の際に許すギャップの幅
Fig. 4 The maximum width of extension

図 5 のように seed 探索と伸長をオーバーラップさせることが可能である．

seed 探索と伸長をオーバーラップさせるために，本研究では GPU に seed を転送するために 2 つのバッファを使用した．まず，マルチ CPU によって seed 探索を行い片方のバッファに seed 探索で得られた seed を貯めていき，ある程度，seed が溜まった段階で GPU にまとめて seed を転送する．そして，GPU は seed の転送が完了次第，伸長を開始し，計算が終わり次第 CPU に結果を転送する．CPU は GPU に seed を転送した時点で，seed を貯めるバッファを切り替え，GPU が処理をしている最中も seed 探索を継続する．

こうすることで，GPU での計算を隠蔽することができる．また，配列マッピングの際にマルチ CPU 対 1 GPU という割合で使用することで 1 GPU あたりの使用効率を上げるこ

Vol.2011-BIO-25 No.32
2011/6/24

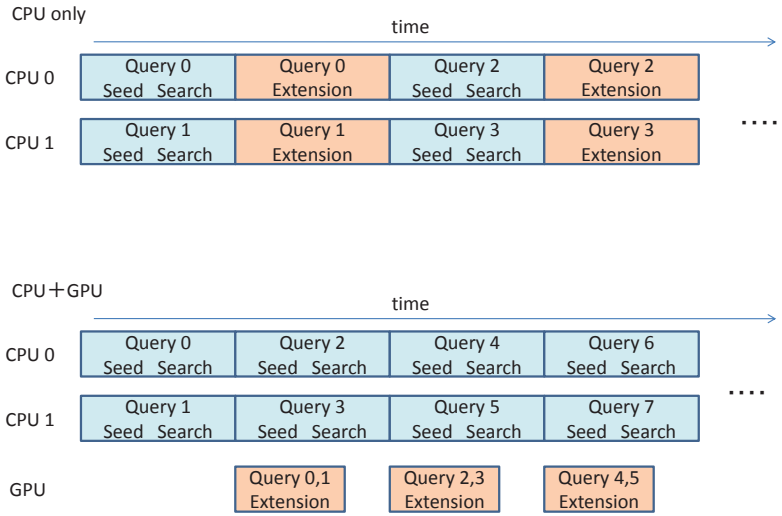

図 5 CPU のみの場合 (上) と GPU を用いた場合 (下) の処理の流れ
Fig. 5 The flow of processing with CPU only (above) and CPU + GPU (below)

とができる．

3. 性能評価実験

本研究で提案した手法の配列マッピングの感度と計算速度を評価する実験を行った．実験に使用した計算機は TSUBAME2 の Thin ノードを使用した．このノードの CPU は Intel Xeon processor X5670 (6 コア, 2.93GHz) を 2 つ，メモリは 54GB，GPU は NVIDIA Tesla M2050 を 3 つである．OS は SUSE Linux Enterprise Server 11 SP1，gcc は version 4.3 であり，GPU を使用するために CUDA3.2 を使用した．性能比較には NCBI BLAST(version 2.2.24) と BLAT(version 34 standalone) を使用した．BLAST はスコア行列に BLOSUM62，open gap penalty と extend gap penalty はそれぞれ 11，1，そしてフィルタを使用しないようにした．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 F -M BLOSUM62” である．BLAT は予め DNA 断片配列をアミノ酸配列に翻訳しておき，それをクエリとして使用した．使用した BLAT のオプションは “-q=prot -t=prot -out=blast8” である．提案手法の seed 探索のパラメータは検索する部分文字列の長さは $L_{max} = 8$ とし，伸長の際のギャップの最大数 $w = 50$ を使用した．また，$T_s = 18, 24, 30$，$D = 1, 4, 7$ の全組み合わせを試し，感度と速度の性能を考慮し，感度重視として $(T_s = 24, D = 4)$，速度重視として $(T_s = 30, D = 1)$ を使用した．その他のパラメータの値は BLAST の値と共通のものを用いた．$D = 1$ のときは seed 探索の際，探す部分文字列は完全一致のペアのみとなる．以下に実験の手順と結果について述べる．

3.1 使用データ

アミノ酸配列データベースは 2010 年 11 月時点の KEGG Genes（genes.pep）[17)–19)] を使用した．このデータベースはタンパク質のアミノ酸配列を 420 万本を含み，全配列の合計長さは 2G 残基である．検索のクエリデータとしては現在の DNA シーケンサーの出力である実データと将来 DNA シーケンサーが改良された際の長い DNA 断片配列出力を想定したシミュレーションデータを 2 つ，合計 3 種類のデータを用いた．実データとして土壌のメタゲノムを illmina 社の Genome analyzer(Solexa) で読み取ったデータを使用した．Solexa のデータは 1 本当たり 60〜75 塩基であり，約 7 百万本からなっている．全データを利用した場合，配列マッピングの感度の比較の際，SSEARCH の計算に多くの時間が必要なため，本研究では全データの内，1 万本をランダムに選択して使用した．また，長い DNA 断片配列のデータとしては DNA シーケンサーのシミュレータである MetaSim[20)] を用いて 2011 年 5 月時点の NCBI の Genomes の Bacteria の全配列から生成した長さ約 500 塩基と約 1000 塩基の DNA 断片配列を 1 万本ずつ準備した．それぞれのクエリデータを L500 と L1000 と呼ぶことにする．

3.2 配列マッピングの感度の比較

配列マッピングの感度を比較するために，DNA 断片配列 1 本毎に配列マッピングの正確さを評価した．厳密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内のアミノ酸配列が最もスコアが高い場合正解とした．クエリには Solexa による実データを使用した．

図 6 に示すように，提案手法の感度重視と BLAST の感度を比較してみると，E-value が 10^{-1} までの領域では BLAST と同等の感度が出ていることが分かる．これは，seed 探索の際に BLAST と同様にいくつかミスマッチを許した部分文字列の探索を行っているためである．ただし，E-value が極端に高い部分では BLAST より感度が低くなってしまっている．これは seed 探索の際に探している部分文字列の長さが BLAST に比べ長く，ミスマッチなどが多く入っているようなものを見つけることができなかったためであると考えられる．

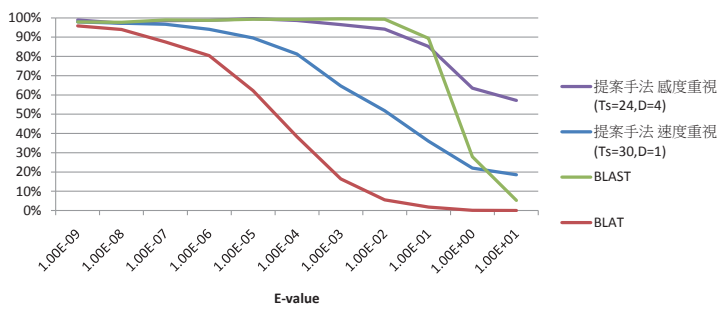

図 6 E-value を変化させた場合の正解が含まれる割合（ % ）
Fig. 6 Correct alignment rate for each E-value（ % ）

また，提案手法の速度重視と BLAT の感度を比較すると，常に BLAT よりも高い感度を保っている．BLAT はデフォルトで長さ 5 の部分文字列の完全一致を検索している．これに対し，提案手法の速度重視では置換行列に BLOSUM62 を使用すると 1 文字一致した場合のスコアの平均が約 6 なので，BLAT と同じく約 5 文字の完全一致を探しているが，スコアが 30 を超えた時点でそれ以上長い部分文字列を検索しないため，実際には BLAT よりも短い部分文字列も検索している．このため，BLAT と同じく完全一致しかみていないにも関わらず，BLAT 以上の感度を保つことができている．

3.3 計算時間の比較

配列マッピングの計算時間を比較するために，Solexa のデータと L500 と L1000 のデータを用いてそれぞれのツールの 1 thread の場合と 12 threads の場合を 5 回計算時間を測定し，計算時間の平均を比較した．提案手法は 1thread のときには 1GPU，12thread のとき 3GPU を使用した場合も測定した．

結果は表 1，表 2 の通りである．

ただし，BLAT の 12 threads に関してはメモリが足りず測定できず，BLAST の 1thread 時の L500 と L1000 に関しては計算機の実行限界時間を超えるため測定できなかった．このため，BLAST の 1 thread 時の L500 と L1000 の計算時間は 12 threads の計算時間から逆算した推定時間である．

また，BLAST の 1 thread を基準にした他のツールの速度向上比を**図 7**，**図 8** に示す．結果としてはクエリの DNA 断片の長さが長くなるにつれて GPU を使っていなかった場合，

表 1 1thread の計算時間 (sec)
Table 1 Computation time of 1 thread(sec)

ツール	Solexa	L500	L1000
提案手法 感度重視 (Ts=24,D=4) 1thread	1,565	16,549	38,273
提案手法 感度重視 (Ts=24,D=4) 1thread + 1GPU	1,711	10,538	22,583
提案手法 速度重視 (Ts=30,D=1) 1thread	329	2,994	7,524
提案手法 速度重視 (Ts=30,D=1) 1thread + 1GPU	274	1,547	2,918
BLAST 1thread	36,641	353,743	749,265
BLAT 1thread	799	6,370	13,240

表 2 12thread の計算時間 (sec)
Table 2 Computation time of 12 threads(sec)

ツール	Solexa	L500	L1000
提案手法 感度重視 (Ts=24,D=4) 12 threads	256	1,784	3,980
提案手法 感度重視 (Ts=24,D=4) 12 threads + 3 GPUs	232	1,544	3,082
提案手法 速度重視 (Ts=30,D=1) 12 threads	105	360	757
提案手法 速度重視 (Ts=30,D=1) 12 threads + 3 GPUs	85	310	557
BLAST 12 threads	3,527	33,855	72,251
BLAT 12 threads	-	-	-

速度向上比が徐々に落ちている場合があるのに対し，GPU を用いた場合，速度向上比が上昇していることが分かる．

1 thread の場合，提案手法は GPU を使っていないと徐々に速度比が低下している．これはクエリの DNA 断片配列が長くなるにつれて伸長の処理に多くの計算時間が必要となるためである．このため，提案手法は GPU を用いた場合，伸長部分を GPU で高速化しているため DNA 断片配列が長くなったとしても速度比は上昇していく．特に，提案手法の速度重視では，seed 探索で完全一致の部分文字列を検索しているため高速であり，伸長の計算時間の割合が多くなっている．このため，提案手法の感度重視の時と比べ GPU を用いた高速化の効果が大きくなっている．

12 threads の場合は seed 探索に多くの時間がかかっている提案手法の感度重視の場合，クエリの DNA 断片配列が長くなるに連れて，徐々に速度比が低下している．これは 1 thread の時と同様に徐々に伸長にかかる計算時間が増加しているためである．このため 12 threads の場合も GPU を用いた場合，seed 探索に時間がかかっていなかった提案手法の速度重視の場合の速度比が大きくなっている．

また，クエリの DNA 断片配列が短い Solexa のデータでは，提案手法の 1 thread の時と

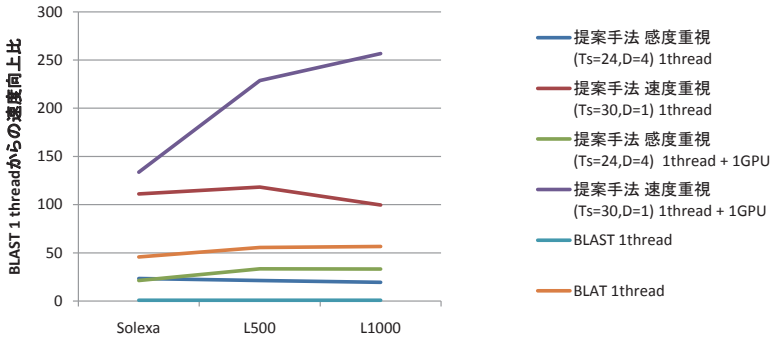

図 7 BLAST を 1 とした場合の計算速度向上比（1 thread）
Fig. 7 Speed up ratio Compared to BLAST(1 thread)

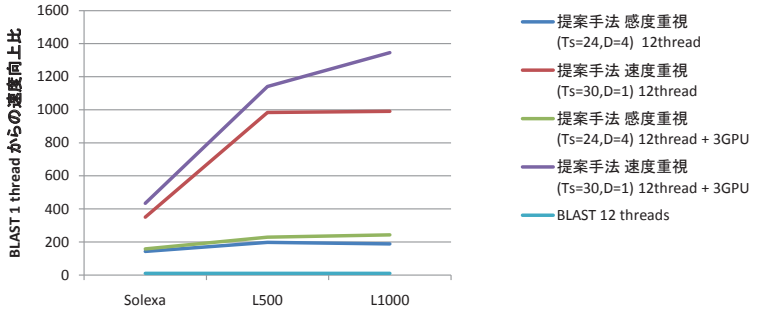

図 8 BLAST を 1 とした場合の計算速度向上比（12 threads）
Fig. 8 Speed up ratio Compared to BLAST(12 threads)

12 threads を比較してみると感度重視の場合は 12 threads の方が 1 thread の時よりも約 6〜7 倍の高速化に留まっているが，クエリの DNA の断片配列が長くなるにつれて徐々に速度比が大きくなり L1000 を用いたとき提案手法の感度重視では約 10 倍の高速化に達している．また，提案手法の速度重視の場合は 12 threads の場合でも感度重視の 12 threads のときより並列化効率がでていない．これはクエリの本数が少ないため，並列化していない部分の計算時間の割合が大きくなり並列化の効果が出ていないと考えられる．これは seed 探索の際，thread 数が増加するとキャッシュミスが多くなり thread 数が多くなってもリニアに速度比が上昇していなかったが，クエリの DNA の断片配列が長くなるにつれて伸長の計算時間の割合が多くなったため，結果として速度比が上昇したと考えられる．

また，提案手法の 1 thread で CPU のみの場合と GPU を用いた場合を比較すると，Solexa の時にはほぼ速度比が等しいのに対してクエリの DNA 断片配列が長くなるにつれて徐々に速度比が上昇し，L1000 に対する速度重視の時には GPU の併用により約 2.5 倍の高速化を達成した．また，12 threads の場合も 1 thread の時と同様にクエリの DNA 断片配列が長くなるにつれて徐々に速度比が上昇し，L1000 に対する速度重視の時，約 1.4 倍の高速化を達成した．結果として提案手法は BLAST の 1 thread と比べると 1 thread の時最大で約 260 倍，12 threads の時には最大で約 1350 倍の高速化を達成た．

4. 結　　論

4.1 本研究の成果

本研究では長い DNA 断片配列でも対応可能な GPU を用いたメタゲノム解析用の配列マッピングの手法を提案し，これを実装した．感度重視の時にはほぼ BLAST と同程度の感度が保たれていた．また，速度重視の時には BLAT 以上の感度を達成した．計算速度はクエリの DNA 断片配列が長くなるにつれて上昇し，結果として提案手法は BLAST の 1 thread と比べると 1 thread の時最大で約 260 倍，12 threads の時には最大で約 1350 倍の高速化を達成した．

4.2 今後の課題

今後，さらに DNA シーケンサーの出力する DNA 断片配列が長くなるとタンパク質のコード領域がすべて DNA 断片配列に収まるようになると考えられる．このため，DNA 断片配列のすべてをアミノ酸に翻訳するのではなく，オープンリーディングフレームを考慮して翻訳したほうが効率がよくなると考えられる．また，現在は配列マッピングの感度を保つためにミスマッチを数文字許した seed 探索を行っているが，この seed 探索をさらに高速化することができれば伸長の計算時間の割合が増加し結果として GPU を用いた高速化の効果が大きくなると考えられる．このため，seed 探索をさらに高速化する必要があると考えられる．

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 a a, et al: “Gapped BLAST and PSI-BLAST: a new generation of protein database search programs”, *Nucleic acids research*, 25(17):3389-3402(1988).
8) Kent WJ: “BLAT–the BLAST-like alignment tool”, *Genome research*,12(4):656-64(2002).
9) Dalevi D, Ivanova NN, Mavromatis K, et al: “Annotation of metagenome short reads using proxygenes”, *Bioinformatics*, 24(16):7-13(2008).
10) 鈴木 脩司 , 石田 貴士, 秋山 泰: “GPU による DNA 断片配列の高速マッピング”, 情報処理学会研究報告バイオ情報学 *(BIO)*, 21(30):1-6(2010).
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) TSUBAME2 — [GSIC] 東京工業大学学術国際情報センター. http://www.gsic.titech.ac.jp/tsubame2
13) Manber, U, Myers, G: “Suffix arrays: A new method for on-line string searches”, *Society for Industrial and Applied Mathematics Philadelphia*, 22(5):935-948(1990).
14) 鈴木　脩司 , 石田 貴士 , 秋山 泰: “FM-index を用いた高速な配列相同性検索ツールの開発”, 情報処理学会研究報告バイオ情報学 *(BIO)*, 23(21), pp. 1-6(2010).
15) Liu Y, Maskell DL, Schmidt B: “CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units” *BMC research notes*, 2:73(2009).
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., and Hirakawa, M.: “KEGG for representation and analysis of molecular networks involving diseases and drugs”, *Nucleic Acids Res*, 38, 355-360(2010).
18) Kanehisa, M., Goto, S., Hattori, M., Aoki-Kinoshita, K.F., Itoh, M., Kawashima, S., Katayama, T., Araki, M., and Hirakawa, M.: “From genomics to chemical genomics: new developments in KEGG”,*Nucleic Acids Res*, 34, 354-357(2006).
19) Kanehisa, M. and Goto, S.: “KEGG: Kyoto Encyclopedia of Genes and Genomes”, *Nucleic Acids Res*, 28, 27-30(2000).
20) Richter DC, Ott F, Auch AF, Schmid R, Huson DH:“MetaSim-A Sequencing Simulator for Genomics and Metagenomics”, *PLoS ONE* 3(10):e3373(2008).