Vol.2013-MPS-96 No.6
Vol.2013-BIO-36 No.6
2013/12/11

De Bruijn Graphの分割によるVelvetの消費メモリの低減

杉浦 典和[1]　石田 貴士[1]　秋山 泰[1]　関嶋 政和[1]

概要：de Bruijn graph を用いる代表的 *de novo* アセンブラである Velvet は，その消費メモリ量の多さが課題とされている．Velvet は大きく 2 つのステップから構成されており，1 つ目のステップについてはハッシュテーブルの分割による消費メモリ量の削減手法が既に提案されている．本稿では後半のステップで Velvet が作成する de Bruijn Graph やその他のデータ構造を分割することで，Velvet の後半の消費メモリ量を削減した．

キーワード：*de novo* アセンブリ， Velvet， 分散処理， de Bruijn Graph， 次世代シークエンサ

Memory cost reduction of Velvet by dividing de Bruijn Graphs

Sugiura Norikazu[1]　Ishida Takashi[1]　AkiyamaYutaka[1]　Sekijima Masakazu[1]

Abstract: It is a well-known fact that the memory consumption of Velvet, which is one of the representative *de novo* assembler based on de Bruijn Graph, is too large. Velvet is composed of two steps, and several methods have been already proposed for decreasing the memory consumption of the first step by dividing the hash table. Here we proposed a graph dividing method. By using this method, we have succeeded to decrease the memory consumption of the latter step of Velvet.

Keywords: *de novo* asesmbly， Velvet， Distrubuted precessing， de Bruijn Graph， Next Generation Sequener

1. はじめに

，

現在，ゲノムの解析には次世代シークエンサと呼ばれるハイスループットなゲノム読み取り装置が用いられている．次世代シークエンサのは，大量のゲノムを高速かつ低価格に読み取ることができる一方，読み取るゲノム断片長配列の長さが数十から数百塩基しかないという欠点を持っている．このため，出力された短いゲノム断片配列をアセンブリすることによって元の長いゲノム配列に再構築しなければならない．

シークエンサから得られるゲノム断片配列をリードと呼び，このリードを他のゲノム配列を参照すること無しに元の長いゲノム配列に再構築することを *de novo* アセンブリと言う．またアセンブリを行うツールのことをアセンブラと呼ぶ．次世代シークエンサの登場以来，短いリードをアセンブリするため，Velvet[1]，Allpaths-LG[2]，SOAPdenovo[3] など様々なショートリード向けのアセンブラが開発されてきた．これらの *de novo* アセンブラは，ショートリードを効率的にアセンブリするために de Bruijn Graph[12] を利用している．これによって次世代シークエンサが出力するような短いリードでも，おおよそ元の長いゲノム配列にアセンブリすることが可能になっている．

しかしこれらのアセンブラはその消費メモリ量が課題とされており，特にヒトゲノムのように大規模なゲノムの *de novo* アセンブリの場合は数テラバイトに及ぶ膨大な量のメモリが必要となることが知られている [4]． *de novo* アセンブリの消費メモリ量を削減するアプローチの一つとして，de Bruijn Graph を分割しクラスタ環境で *de novo* アセンブリを行う手法がある．ABySS[5]，YAGA[6]，PASHA[7]

[1] 東京工業大学　大学院情報理工学研究科
Tokyo Institute of Technology, Meguro, Tokyo 152-8550. Japan

などのアセンブラは，クラスタ間で MPI 通信を行うことで *de novo* アセンブリを行うことができる．しかしこれらのアセンブラは計算ノード数が増えるに従い非線形に増加する高い通信コストが課題とされている．*de novo* アセンブリの消費メモリ量を削減する別のアプローチとして，de Bruijn Graph の構造を最適化することで消費メモリ量を削減する手法も提案されている．SparseAssembler[8] は k-mer の一部のみを保存することでシーケンシングエラーによる de Bruijn Graph の複雑化を抑制し，消費メモリ量を劇的に削減した．また SGA[9] はリードの圧縮インデックスを用いることで de Bruijn Graph の消費メモリ量を削減し，Human genome を 60GB 以下でアセンブリ可能にした．しかしこれらのアセンブラでも，現在の市販のコンピュータで実行するにはまだ消費メモリ量が大きい．

本研究では de Bruijn Graph を分割し，各分割グラフ間で通信を行わずに *de novo* アセンブリ行う手法を提案する．この手法により，搭載メモリの少ない単一の計算機でもグラフの分割数を増やすことで大規模なゲノムの *de novo* アセンブリが実現できるようになる．ここでは，代表的なショートリード向けのアセンブラの 1 つであり，アセンブリ結果に一定の信頼が得られている Velvet を改良し，Velvet のアセンブリ結果を維持したまま消費メモリ量を削減することを目指す．

Velvet は前半と後半の 2 ステップから構成されており，一つ目は velveth，二つ目は velvetg と呼ばれている．velveth はハッシュテーブルを利用することで，リードのオーバーラップの情報を記録した ROADMAPS というファイルを作成しており，この過程はハッシュテーブルを分割することで既に分散化が実装されている [10][11]．これに対し velvetg は ROADMAPS に書かれた情報を元にして de Bruijn Graph を作成しアセンブリを行う．velvetg では，はじめに PreGraph と呼ばれる de Bruijn Graph を作成し，この Graph を元にして再度 de Bruijn Graph を作成している．これまで PreGraph の分割に関する研究は行われてきたが [13]，velvetg 全体の分散化はまだ成されていない．本研究では，PreGraph の作成ステップから Graph の生成ステップ，paired-end リードを利用した scaffold と呼ばれる非常に長いコンティグの生成ステップまでを分散化することで，少量のメモリしか搭載していない 1 台の計算機でも大きなゲノムをアセンブリできるように改良することを目指した．

2. Velvet の構成

Velvet は二つの実行ステップから構成されており，一つ目は vlveth，二つ目は velvetg と呼ばれている．vevleth はハッシュテーブルを利用することで入力リード中の一致する k-mer を検索し，リードのオーバーラップ情報を記録した「ROADMAPS」と呼ばれるファイルを作成する．velvetg では velveth で生成した ROADMAPS を参考にして，初期の de Bruijn Graph である PreGraph を生成する．PreGraph に対しエラー削除アルゴリズムを適用しシーケンスエラーに起因するノード等を切り捨てた後，改めて de Bruijn Graph を生成する．この de Bruijn Graph に対し，さらに厳しいエラー削除アルゴリズムや，Paired end リードを利用した Scaffold の作成等を行い，最終的なコンティグを作成する．

velvetg を実行した際の消費メモリ量の推移を図 1 に示す．図 1 における各ステップ番号は以下に対応する．

1 PreGraph の生成
2 入力リードの読み込み
3 k-mer 検索テーブルの作成と Graph のエッジの生成
4 TourBus アルゴリズム等のエラー削除
5 Paired-end の挿入長等を用いた scaffold の生成

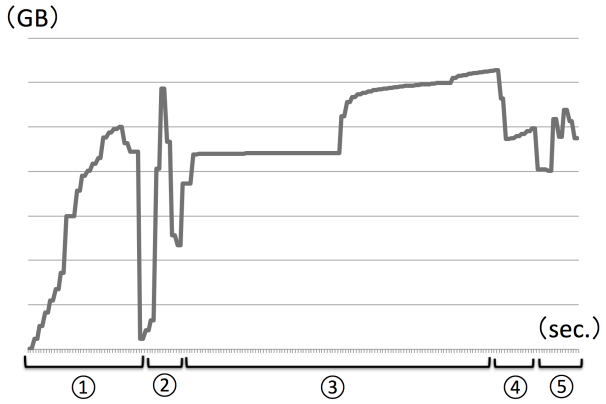

図 1 velvetg の消費メモリ量の推移

本研究では velvetg の消費メモリ量低減を行うため，velvetg について詳しい処理の流れを以下に記す．

2.1 PreGraph の生成

velvetg では，まず最初に PreGarph と呼ばれる初期のグラフを生成する．PreGraph は以下のステップで作成される． まず velveth で作成した ROADMAPS を全て主記憶に読み込み，作成する PreGraph のバーテックス数を数える．このとき各バーテッ クスがどのリードから生成されるかという情報を，主記憶上の ROADMAPS に新たに書き込む． 次にこれらの情報に基づきバーテックスとエッジを作成し，最後にシーケンシングエラーを削除するために短い dead-end パスの削除や一続きのパスの結合を行う．このようにして PreGraph を作成する．PreGraph が完成したら，そのノードのみをファイルに書き出す．

2.2 入力リードの読み込み

Velvet は入力リードを圧縮して保存している．塩基は ACGT(又は N) の 4 塩基なので，各塩基は 2 ビットで表

現可能である (N は A に置き換える)．Velvet はリードの読込みに置いて，まず入力リードを非圧縮のまま全てメモリに読み込み，次にそれらを全て圧縮する．圧縮が終わったら，始めに読込んだ非圧縮の入力リードを解放する．このため図 1 において，入力リードのに読み込みステップでピークメモリが大きくなっている．

2.3 Graph の生成，エラー削除

Velvet において，Graph のノードは PreGraph のノードをそのまま引き継いでいる．そのため，あとはエッジを生成すれば Graph の基本構造は完成する．エッジを生成し初期のグラフが完成したら，これに複数のエラー削除アルゴリズムを適応することで Graph の構造を単純化し，最終的な Graph として完成する．Graph 生成は以下の手順で行う．

Step 1 PreGraph から全てのノードを読込む

Step 2 各ノードから一塩基ずつずらしながら k-mer を読み込み，その k-mer のノード ID とノード内の位置情報を添えてテーブル (k-mer 検索テーブル) に格納 (図 2)

Step 3 k-mer を格納し終えたら，後の二分探索のために k-mer 検索テーブルを k-mer の値についてソート

Step 4 以下を全ての入力リードについて行う．

- リードセットから k-mer を一本ずつ取り出し，その k-mer を k-mer 検索テーブルで二分探索しノード ID を取得
- リードセット内の隣り合う k-mer に別々のノード ID が添付されていた場合，これらのノードは隣り合っていることが分かるので，これを結ぶエッジを生成 (図 3)

Step 5 TourBus アルゴリズムや dead-end パスの削除などを行い，Graph の構造を単純化．

k-mer	NodeID	位置
AAAACTTACTTCCTTAT	1	0
AAACTTACTTCCTTATT	1	1
AACTTACTTCCTTATTT	1	2
ACTTACTTCCTTATTTG	1	3
CTTACTTCCTTATTTGG	1	4

図 2 k-mer 検索テーブルの生成

2.4 Scaffold の生成

入力するリードが Paired-end リードの場合，その挿入長などを利用することでより長いコンティグ (Scaffold) を生成することができる．Scaffold の生成は，以下の手順で行う．

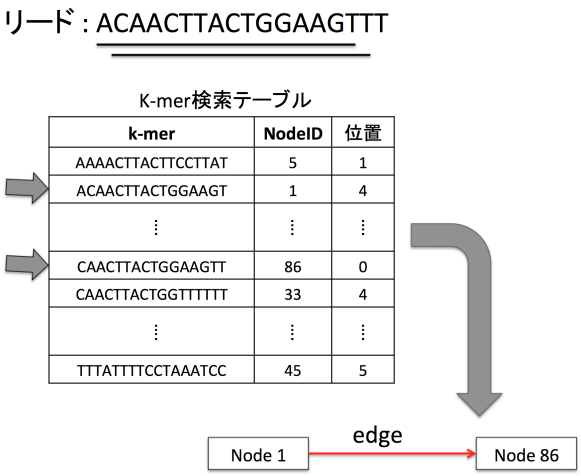

図 3 エッジの生成

Step 1 各ノードについて，そのノードを構成するリード ID を取得 (この処理は，Graph 生成時に行っておく)

Step 2 各リードについて，そのリードから得られるノード ID を取得 (Step 1 の情報から逆算する)

Step 3 Paired read 情報と Step 2 の情報を用いて，対応するノードペアを取得 (図 4)

Step 4 ペアになっているノード (またはペアのペアになっているノード) が隣接しているとき，それらのノードを接続

このようにして得られた Graph に対し，再度 TourBus アルゴリズムや dead-end のパスの削除などのエラー削除を行い，最終的なグラフを作成する．

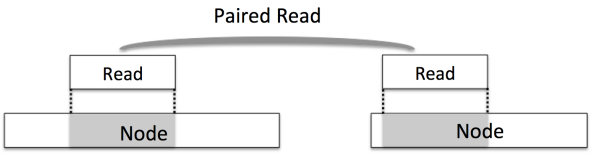

図 4 対応するノードペアの取得

3. 消費メモリ量削減の提案手法

ここでは，Velvet の後半の処理である velvetg の消費メモリ量の削減手法を提案する．velvetg は，de Bruijn Graph そのもの以外にも，de Bruijn Graph の構築・最適化を行うための複数のデータ構造を利用している．このため velvetg の消費メモリ量を削減するためには，これらのデータ構造の消費メモリ量を削減する必要がある．本研究では特に消費メモリ量の著しい三つのデータ構造に着目し，この消費メモリ量を削減する．一つ目は PreGraph，二つ目は k-mer 検索テーブル，三つ目は Read-Node 関係テーブルである．またこれらのデータ構造のほか，入力リードセットもピー

クメモリを押し上げる要因の一つとなっている．本章では各データ構造の分割手法について述べる．

3.1 PreGraph の分割

PreGraph の生成ステップにおいては，PreGraph そのものよりも PreGraph を生成するために利用する ROADMAPS がより大きなメモリを消費する．本研究では PreGraph をノード数が均等になるように分割することで，その分割 PreGraph の生成に必要となる ROADMAPS のサイズも縮小し，PreGraph 生成ステップにおける消費メモリ量を削減した [13].

3.2 リードセットの読み込み

リードセットを読込む際は，一度全てのリードを文字列としてメモリに保存してから圧縮しているため，ピークメモリが高くなっている (図 1)．そこで，リードを文字列としてメモリに保存する手順を省略し，ファイルから読込むと同時に文字列を圧縮し，メモリに圧縮リードセットを保存するように再実装した．また一度読込んだ圧縮リードセットを最後までメモリに保持するのではなく，リードを扱う必要があるステップに入るときのみ圧縮リードセットを読込むようにすることで，より消費メモリ量を低減した．さらにリードを扱うステップでも，圧縮リードセットを全て読込むのではなく，圧縮リードセットを複数の分割圧縮リードセットに分割して読込ことで，他のデータ構造の分割数に応じてリードセットのサイズも任意に低減できるように実装た．

3.3 k-mer 検索テーブルの分割

Graph の生成ステップでは，エッジの生成ステップで利用する k-mer 検索テーブルなどがピークメモリの原因となっている．ここでは k-mer 検索テーブルを複数に分割しメモリに順次に読込むことで，消費メモリ量を削減する．

k-mer 検索テーブルは PreGraph から読込んだノードから，長さ k のワードを取り出し格納したテーブルである．ここでは PreGraph のノード数を均等に分割し，各ノードセットにおいて分割 k-mer 検索テーブルを作成するように実装した (図 5)．分割 k-mer 検索テーブルとリードセットを参照してエッジを生成する手法は Velvet と同様である．k-mer 検索テーブルに比べエッジの消費メモリ量は小さいため，現在の実装ではエッジを一括してメモリに保持しているが，今後エッジも複数に分割する予定である．

3.4 Read-Node 関係テーブルの分割

scaffold の生成ステップでは，Paired-end リードに対応するノード同士を選択するため，あるリードから生成されるノードのリスト (Read-Node 関係テーブル) を作成しており，このテーブルがメモリを消費している．そこで

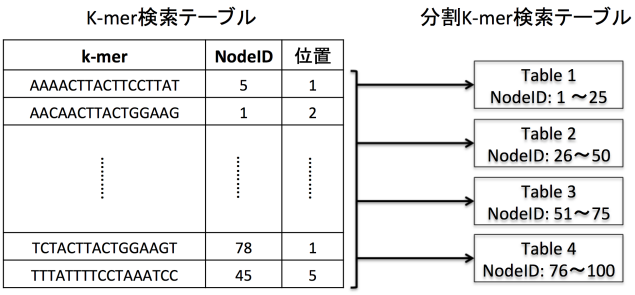

図 5 k-mer 検索テーブルの分割 (ノード 100 本の場合)

表 1 Staphylococcus のリードセット

	Flagment	Short jump library
平均リード長	101 bp	37 bp
リード本数	1,294,104	3,494,070
挿入長	180 bp	3500 bp
シークエンサ	Illumina GA II	

表 2 Human Chromosome 14 のリードセット

	Flagment	Short jump library	Long jump library
平均リード長	101 bp	101 bp	76 - 101 bp
リード本数	36,504,800	22,669,408	2,405,064
挿入長	155bp	2283 - 2803 bp	35,295 - 35,318 bp
シークエンサ	Illumina HiSeq 2000		

Read-Node 関係テーブルを分割し，特定のリードについてのみ関係テーブルを作成することにより，消費メモリ量を削減した．なおテーブルの分割にあたっては，各リードが持つ全てのリードリストの要素数をメモリの確保前に算出し，各 Read-Node 関係テーブルがもつリスト要素の数が等しくなるように分割した．

4. 実験

4.1 データセット

実験に用いるデータセットには，Stapylococcus と Human Choromosome 14 のリードセットを用いた．どちらも GAGE[14] のウェブサイトからダウンロードした．各リードセットの詳細を表 1，表 2 に示す．

4.2 実行環境・測定方法

実験には，東京工業大学が所有する計算機，TSUBAME 2.5 を使用した．使用した計算ノードの詳細を表 3 に示す．Staphylococcus のリードセットを用いる実験は計算ノード S で，Human Chromosome 14 のリードセットを用いる実験は計算ノード S96 で行った．

分散化によってファイル入出力の回数が増えている (主に PreGraph の分割処理) ため，本実験では計算ノードに搭載された SSD を使用した．また実験中に，書き出した

表 3 計算機環境

	S	S96
CPU	Intel Xeon 2.93 GHz (6 cores) x 2	
メモリ	54GB	96GB
SSD	120GB	240GB

ファイルサイズが SSD の容量を超えることは無かった．

消費メモリ量と実行時間の測定のため，ps -l コマンドを一秒おきに実行するシェルスクリプトを作成し，コマンドが返す消費メモリ量を記録した．また velvetg プロセスが存在する時間から実行時間を求めた．

4.3 実行時のパラメータ

Vevlet は実行時のパラメータとして k 値を指定する必要がある．本実験では，Staphylococcus のリードセットを用いる実験は k=35 で，Human Chromosome 14 のリードセットを用いる実験は k=51 で行った．

Paired-end リードにおいては挿入長と挿入長分散を指定する必要があるが，挿入長分散はリードセットに記されていなかったため，挿入長の約 1 割として入力した．

Velvet で Paired-end リードをアセンブリする場合，リードセットは innie で無ければならない．Stapylococcus の Short jump library と Human Chromosome 14 の Short jump library は outie だったので，FASTX-Toolkit[15] にて innie に変換した上で Velvet に入力した．

また Graph の生成においては，k-mer 検索テーブルと Read-Node 関係テーブルの二種類のテーブルを分割するが，ここでは Read-Node 関係テーブルの分割数を k-mer 検索テーブルの分割数の二倍として実験を行った．これは実験結果の図 9 などを参考にし，Read-Node 関係テーブルのサイズが k-mer 検索テーブルのサイズを超えないよう経験的に得られた値である．

4.4 実験結果

まず Staphylococcus のリードセットを入力とした場合について，PreGraph を分割しない場合と分割する場合の消費メモリ量と実行時間をそれぞれ比較する．このとき生成された PreGraph は全て一致することを確認している．PreGraph の分割数を 4 から 128 まで増やした場合の，消費メモリ量と実行時間を表 4 に，消費メモリ量の推移を図 6 に示す．図 6 より，PreGraph の分割数が増えるに従いピークメモリが減少していることが確認できる．また分割数を 32 分割以上に増やしても消費メモリ量の減少は僅かであり，減少幅が頭打ちになっていることがわかる．128 分割に分割したときの消費メモリ量は，改良前の消費メモリ量に対し 68%削減できている．

次に Staphylococcus のリードセットを入力とした場合について，k-mer 検索テーブル及び Read-Node 関係テーブルを分割しない場合と分割する場合の消費メモリ量と実行時間をそれぞれ比較する．このとき生成された Graph やコンティグは全て一致することを確認している．k-mer 検索テーブルの分割数を 4 から 16 まで増やした場合の消費メモリ量と実行時間を表 5 に，消費メモリ量の推移を図 7 に示す．図 7 より，Graph の生成ステップにおいても分割数が増えるに従いピークメモリが減少していることが確認できる．また文分割数を 8 分割以上に増やしても消費メモリ量の減少は僅かであり，減少が頭打ちになっていることが読み取れる．16 分割に分割したときの消費メモリ量は，改良前の消費メモリ量に対し 56%削減できている．

表 4 PreGraph 生成ステップの計算コスト (Staphylococcus)

	消費メモリ量 (MB)	実行時間 (sec)
非分割	366	19
4 分割	230	159
8 分割	206	233
16 分割	156	366
32 分割	127	627
64 分割	121	1135
128 分割	115	2140

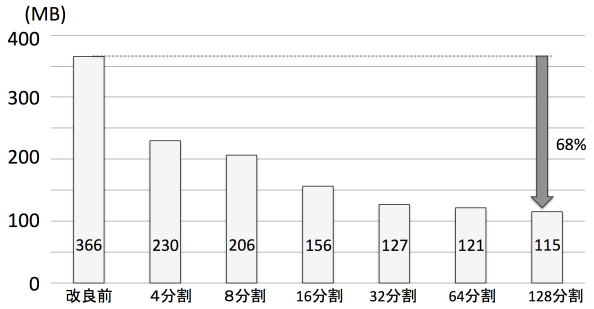

図 6 PreGarph の各分割数に対する消費メモリ量

表 5 Graph 生成ステップ等の計算コスト (Staphylococcus)

	消費メモリ量 (MB)	実行時間 (sec)
改良前	607	61
効率化後	417	120
4 分割	359	354
8 分割	269	1223
16 分割	266	4645

また Staphylococcus のリードセットを入力とした場合について，PreGarph を 4 分割したときの消費メモリ量の推移を図 8 に，k-mer 検索テーブルを 4 分割，Read-Node 関係テーブルを 8 分割したときの消費メモリ量の推移を図 9 に示す．

次に Human Choromosome 14 のリードセットを入力とした場合について，PreGraph 及び k-mer 検索テーブル等を分割しない場合と分割する場合の消費メモリ量と実行時間をそれぞれ比較する．Human Choromosome 14 の入力リードに対しては，PreGraph や k-mer 検索テーブルを 4

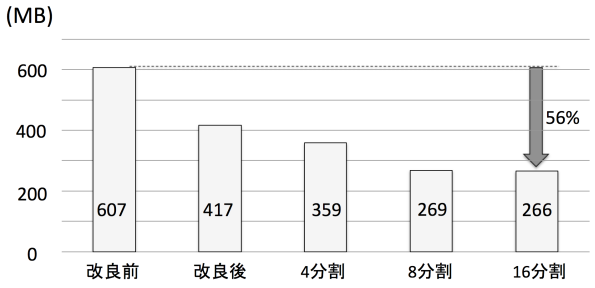

図 7 Garph の各分割数に対する消費メモリ量

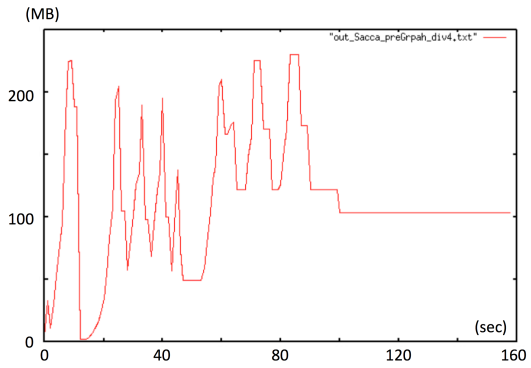

図 8 PreGraph 生成ステップを 4 分割したときの消費メモリ量の推移

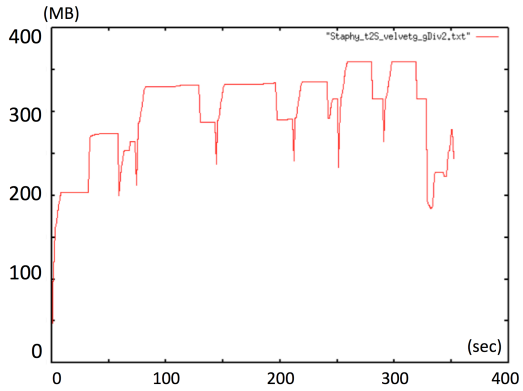

図 9 Graph 生成ステップを 4 分割したときの消費メモリ量の推移

分割する場合についてのみ求めた．消費メモリ量削減前と削減後のピークメモリの対応表を表 6 に，実行時間の対応表を表 7 示す．表 6 より，PreGraph 生成ステップの消費メモリ量は約 41%，Graph 生成ステップの消費メモリ量は約 59%削減されていることが分かる．

ただしこのとき生成された Graph 及びコンティグは，分割しない場合の結果すなわち Velvet 本来の結果と比べ若干の誤差があった．詳細は考察で述べる．

表 6 Human Chromosome 14 における消費メモリ量

	PreGraph	Graph
オリジナル	9,469 MB	11,476 MB
4 分割 ver.	5,649 MB	4,746 MB

表 7 Human Chromosome 14 における実行時間

	PreGraph	Graph
オリジナル	533 sec	3,262 sec
4 分割 ver.	5,424 sec	16,792 sec

5. 考察

図 6，図 7 より PreGraph 生成過程と Graph 生成過程の両方において，分割数を増やすとともに消費メモリ量が減少していることがわかる．しかしどちらも線形には減少していない．これは PreGraph や k-mer 検索テーブの他にもメモリを消費しているデータ構造が複数あるためであり，それらを分割していないことが原因と考えられる．

次に Human Choromosome 14 のリードセットを入力とした実験で結果が完全には一致しなかったことについて述べる．Velvet は PreGraph から Graph を生成しエラー削除を行うと，一度 Graph を完成とみなし「Graph」ファイルに書き出す．その後その Graph に対し Paired-end リードの情報を用いて Graph を単純化し，単純化された最終的な Graph を「LastGraph」としてファイルに出力する．最後に Graph に含まれるノードで一定の基準を満たすノードのみを「contig」ファイルに出力し終了する．

本実験で一致しなかったファイルは LastGraph と contig であり，Graph は一致していた．そのため Graph から LastGraph を生成する過程，すなわち paired-end リード情報を用いて Graph を単純化するステップに原因があると予想される．Velvet が生成した LastGrpah は 85758 本のノードを持っていたのに対し，Read-Node 関係テーブルを分割して生成した LastGraph は 85819 本のノードを持っており，ノードが 61 本多くなっていた．paired-end リード情報を用いて Graph を単純化するステップは，ノード同士を接続することで Graph を単純化するステップなので，本来接続されるべき 61 本のノードが接続されていなかったと考えられる．

6. 結論

PreGraph や k-mer 検索テーブル等を複数に分割することで，velvetg の消費メモリ量の削減に成功した．その結果 PreGraph 生成ステップで最大およそ 68%(Staphylococcus のリードセットに対し 128 分割時)Graph 生成ステップで最大およそ 61%(Human Choromosome 14 のリードセットに対し 4 分割時) の消費メモリ量を削減できた．

7. 今後の課題

本研究では velvetg の消費メモリ量の低減を実現したが，一部のリードセットでは出力結果の不一致が確認されている．Velvet と同等の精度を出せるよう，出力結果を完全に一致させたい．また今回の提案手法は，本来 Velvet がサ

ポートしている機能であるリファレンスシーケンスの入力やロングリードのアセンブリには対応していない．今後はこれらについても改良し，幅広い種類の入力に対応させることを目指す．

参考文献

[1] Daniel R. Zerbino and Ewan Birney: Velvet: Algorithms for *de novo* short read assembly using de Bruijn graphs, *Genome Res.* 18: 821-829, (2008)
[2] Sante Gnerre, Iain MacCallum, Dariusz Przybylski, Filipe J. Ribeiro, Joshua N. Burton, Bruce J. Walker, et al. : High-quality draft assemblies of mammalian genomes from massively parallel sequence data, Proc Natl Acad Sci U S A, 108, 1513-1518.(2011)
[3] Li, R., Hongmei Zhu, Jue Ruan, Wubin Qian, Xiaodong Fang, et al. : De novo assembly of human genomes with massively parallel short read sequencing, Genome Research, 20, 265-272. (2010)
[4] Schatz, Michael: Assembly of Large Genomes using Cloud Computing, http://schatzlab.cshl.edu/presentations/2010-07-23.Illumina.pdf, (2010)
[5] Jared T. Simpson, Kim Wong, Shaun D. Jackman, Jacqueline E. Schein, Steven J.M. Jones, and Inanxc Birol : ABySS: A parallel assembler for short read sequence data, *Genome Res.* 19: 1117-1123,(2009)
[6] Jackson BG, Regennitter M, Yang X, Schnable PS, Aluru S: Parallel de novo assembly of large genomes from high-throughput short reads. Parallel & Distributed Processing (IPDPS), IEEE International Symposium on 1-10. (2010)
[7] Liu Y, Schmidt B, Maskell D: Parallelized short read assembly of large genomes using de Bruijn graphs. BMC Bioinformatics, 12(1):354.(2010)
[8] Chengxi Ye, Zhanshan Sam Ma, Charles H Cannon, Mihai Pop, Douglas W Yu : Exploiting sparseness in de novo genome assembly. BMC Bioinformatics, 13(Suppl 6):S1, (2012)
[9] Simpson, J.T. and Durbin, R : Efficient de novo assembly of large genomes using compressed data structures, Genome Research, 22, 549-556.(2012)
[10] 宇治橋善史, 成瀬彰, 宮本青, 重元康, 北館智：de novo assembler Velvet の メモリ使用量を削減するプロセス並列手法, *IPSJ SIG technical reports* 2012-BIO-28(9), 1-6, 2012-03-21,(2012)
[11] 杉浦 典和, 石田 貴士, 関嶋 政和, 秋山 泰: ハッシュテーブルの分割による de novo アセンブリの改良, *IPSJ SIG Technical Report.* (2012)
[12] Pevzner, P.A., Tang, H. and Waternam, M.S.: An Eulerian path approach to DNA fragment assembly, *Proc. Natl Acad. Sci. USA*, Vol.14, pp.9748-9753.
[13] 杉浦 典和, 石田 貴士, 秋山 泰, 関嶋 政和: De Bruijn graph の PreGraph の分割による Velvet の改良, *IPSJ SIG Technical Report.* (2012)
[14] Salzberg SL, Phillippy AM, Zimin AV, Puiu D, Magoc T, Koren S, Treangen T, Schatz MC, Delcher AL, Roberts M, et al.: GAGE: A critical evaluation of genome assemblies and assembly algorithms. *Genome Res*, (2011)
[15] Gordon A, Hannon GJ “ FASTX-Toolkit ”, FASTQ/A short-reads pre-processing tools (unpublished) http://hannonlab.cshl.edu/fastx_toolkit/.