Vol.2012-BIO-28 No.9
2012/3/28

de novo assembler Velvet の メモリ使用量を削減するプロセス並列手法

宇治橋 善史[†1] 成瀬 彰[†1] 宮本 青[†2]
重元 康昌[†2] 北館 智[†2]

Velvet は次世代シーケンサ装置から出力される短 Read 配列のアセンブルを得意とし，広く利用されている de novo アセンブラである．しかし，Velvet は処理に多くのメモリを消費するため，長大な DNA 配列を持つ生物種の処理には大容量メモリマシンを用意する必要がある．我々は，この問題を解決するために，Velvet のプロセス並列化を実施した．本稿では，Velvet の 1 構成プログラムである velveth のプロセス並列化手法を提案し，提案手法の実装・評価を行い，提案手法の効果を確認した．この手法により，標準的な PC クラスタシステムで大量配列アセンブルが可能になる．

Process parallelization for reducing memory usage of de-novo assembler Velvet

Yoshifumi Ujibashi ,[†1] Akira Naruse ,[†1]
Sei Miyamoto ,[†2] Yasumasa Shigemoto [†2]
and Satoshi Kitadate[†2]

Velvet is a de novo assembler for short reads from next generation sequencer and is broadly used. Velvet consumes large amount of memory, so it needs large-scale memory machine to assemble sequences of spieces with large scale chromosomes. In order to solve this problem, we addressed the parallelization of velvet. In this argicle, we propose the method for process parallelization of velveth which is a component program of Velvet, and confirm its effectiveness by implementation and estimation of the proposed method. Our method realizes assembling of large amount of reads on typical PC cluter systems.

1. はじめに

生物遺伝子情報を司る DNA 配列情報はゲノム研究にとって必要不可欠な基盤情報であり，DNA 配列決定 (シーケンシング) 技術はゲノム研究にとって最も重要な技術のひとつである．

シーケンシングを実行するシーケンサー装置は，数年前に登場した次世代シーケンサーで飛躍的に性能向上した．約十数年前に数百台のシーケンサーで数年かけて決定したヒト DNA 配列 (約 30 億塩基対) を，現在では次世代シーケンサーにより，ヒト DNA 配列を 1 台で約 1 日でシーケンスすることが可能となるまで処理速度が向上している．それに加えて装置・ランニングコストの減少，自動化などによりシーケンス装置が広く普及するようになり，ますます重要な技術になってきている．

一方で，シーケンサー装置は，長大な DNA 配列を直接決定することはできない．よって，シーケンサー装置はまず超音波処理でランダムな位置で DNA 配列を断片化し，それらの断片化した DNA 配列 (Read 配列) の配列を決定する．よって，元の全 DNA 配列を得るためには，各 Read 配列のオーバーラップ配列部分を糊代にしてつなぎ合わせ，再構成する作業を行なう必要がある．この Read 配列をつなぎ合わせる処理を行なうアプリケーションをアセンブラと呼ぶ．そのなかでも既存リファレンス配列を必要とせず，Read 配列情報のみから新規ターゲットを再構築するアプリケーションを de novo アセンブラと呼ぶ．

de novo アセンブラはベースとするアルゴリズムにより 2 種類に大別できる．

ひとつは，Overlap/Layout/Consensus アルゴリズムを採用するアセンブラであり，Celera[8]，Arachne[1]，PCAP[5] などがある．この方式は，ペアワイズアライメントによって Read 配列のオーバーラップを見つけて Read 配列同士をつなぎ合わせる方法で，従来から採用されてきた方法である．しかし，最近主流になった次世代シーケンサーが出力する Read 配列は，各 Read 配列は非常に短い（数十 bp 程度）特徴があり，Overlap/Layout/Counsensus アルゴリズムはこのような超短配列の処理は不得手である．

もうひとつは，de Bruijn Graph[9] を利用するアルゴリズム (de Bruijn Graph 方式) を採用するアセンブラであり，Velvet[12]，ABySS[11], AllPaths[3], Ray[2]，SOAPdenovo[10] な

†1 株式会社富士通研究所
FUJITSU LABORATORIES LTD.
†2 富士通株式会社
FUJITSU LIMITED

どがある[7]．de Bruijn Graph 方式は，次世代シーケンサの短 Read 配列をアセンブルする処理を得意とするアルゴリズムで，近年この方式を採用するアセンブラが主流となっている．

本稿では，このなかでも最も広く利用されている Velvet に着目する．Velvet は短い Read 配列の処理に向いているが，一方で，入力の Read 配列数が膨大になると，メモリを大量に消費する問題がある．次世代シーケンサが出力する通常の Read 配列のアセンブルを実行すると，数十〜数百 GB のメモリを消費することも珍しくなく，実行のために高価な大容量メモリ計算機を必要とする問題がある．

これを解決するため，本稿では，Velvet を構成する velveth/velvetg プログラムのうちメモリ消費量が多い velveth のプロセス並列化手法を提案する．これにより，velveth の 1 プロセスあたりの使用メモリ量を削減し，分散メモリ環境で実行できるようにすることで，大容量メモリ計算機が必要としなくなる．

本稿では，提案手法で velveth のプロセス並列処理を実装し，逐次，2 並列，4 並列，8 並列の場合のプロセスあたりメモリ使用量を比較した．その結果，逐次と比較して，プロセス 4 並列で 1/3 以下，プロセス 8 並列で 1/6 以下のメモリ量消費ですむことを確認した．

以下，2 章で，Velvet の特徴とメモリ消費量の問題点を述べ，3 章で，我々が提案する velveth のプロセス並列化手法を説明し，4 章で，提案手法に基づいてどのように実装したかを述べ，5 章で，実装したプロセス並列化 velveth を評価し，6 章で，まとめと今後の課題を述べる．

2. Velvet の特徴と課題

2.1 概　　要

Velvet は，de Bruijn Graph 方式を採用したアセンブラである．de Bruijn Graph 方式では，まず Read 配列から順次固定長の断片（K-mer 断片）を取り出す．K-mer 断片をノードとし，K-mer 断片のオーバーラップ部分を辺で接続し，de Bruijn Graph を構築する．そして，de Brujin Graph のすべての辺を通る経路を求める Euler 経路問題を解くことで，配列を再構成，元の DNA 配列を決定する方式である．

次世代シーケンサは短かく膨大な数の Read 配列を出力する．de Bruijn Graph は，K-mer 断片のオーバーラップ部分をつなぎ合わせて構成するので，原理的に次世代シーケンサが出力する短い Read 配列を扱うのに適しており，次世代シーケンサが出力するデータをアセンブルする方式として広く採用されるようになった．

2.2 課　　題

しかし，前述のように，Velvet は実行のために必要なメモリサイズが非常に大きい問題がある．文献 6) では，ヒト染色体のなかで比較的小さいサイズの 20 番染色体のアセンブルに 64GB のメモリが必要であると報告している．他のヒト染色体の長さは 4〜5 倍あるので，それらのヒト染色体や他の哺乳類染色体の Read 配列データをアセンブルするためには数 100GB オーダのメモリ量が必要となる．また，文献 4) では，開発元の EMBL 技術者が，通常 Velvet は数十から数百 GB のメモリを消費し，必要メモリが 1TB に達することもあると報告している．

よって，次世代シーケンサが出力した，比較的大きいゲノムサイズの生物の Read 配列をアセンブルするためには，数百 GB 以上のメモリを搭載した高価な大容量メモリ計算機を用意する必要があり，Velvet 利用の際のボトルネックになっている．

次に，Velvet の，どの処理がメモリボトルネックかを述べる．Velvet は，velveth と velvetg の 2 プログラムで構成されており，各プログラムの処理内容は以下の通りである．

- velveth
  重複して出現する K-mer が，どの K-mer と重複し，どの箇所に出現するかを示す K-mer インデックステーブルを作成する
- velvetg
  K-mer インデックステーブルを用いて de Bruijn Graph を生成，更新．読み取りエラー除去，繰り返し配列の解決．Euler 経路を求める．

図 1 に，Read 長が 100，K-mer 長が 31 で実行したときの velveth と velvetg のメモリ使用量をしめす．図 1 からわかるように，velveth，velvetg ともに，Read 配列の数の増加に従って使用メモリ量が増加するので，大量 Read 配列を処理する際にメモリが処理のネックとなる．また，velveth と velvetg を比較すると velveth のほうが使用メモリ量が多いので，Read 配列が増加すると velveth がメモリネックになる．そこで本稿では，veveth のメモリ消費量削減に着目し，velveth のプロセス並列化手法を提案する．

3. velveth のプロセス並列化手法の提案

3.1 K-mer インデックス処理

velveth は前述のとおり K-mer インデックステーブルを出力する．K-mer インデックステーブルは，図 2 のように，重複して出現する K-mer がどの Read 配列のどの場所に存在するか（出現箇所）と，重複する K-mer の出現箇所を示すテーブルである．この K-mer イ

Vol.2012-BIO-28 No.9
2012/3/28

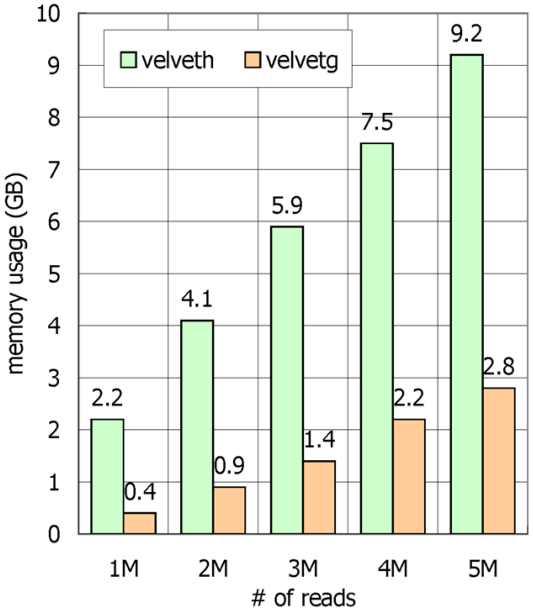

図 1 Velvet メモリ使用量
Fig. 1 Memory usage of Velvet

ンデックステーブルを作成するために，velveth は以下の処理をする．

(1) Read 配列を順番に入力する．
(2) Read 配列から指定された長さの K-mer を順次取り出す．
(3) 各 K-mer 値を key にして入力 Read 配列のどこに存在するか（出現箇所）をハッシュテーブルに保存していく．
(4) その K-mer 値が既にハッシュテーブルに登録されていたら，K-mer の出現箇所を K-mer インデックステーブルに出力する．

この処理を，入力される全 Read 配列に対して行うことで，重複する K-mer の出現箇所を示すインデックステーブルが完成する．以下，この処理を K-mer インデックス処理と呼ぶ．

3.2 提案手法

本節では， 我々が提案するプロセス並列化手法を説明する．

提案手法のポイントは，K-mer インデックス処理において，各プロセスが担当する K-mer

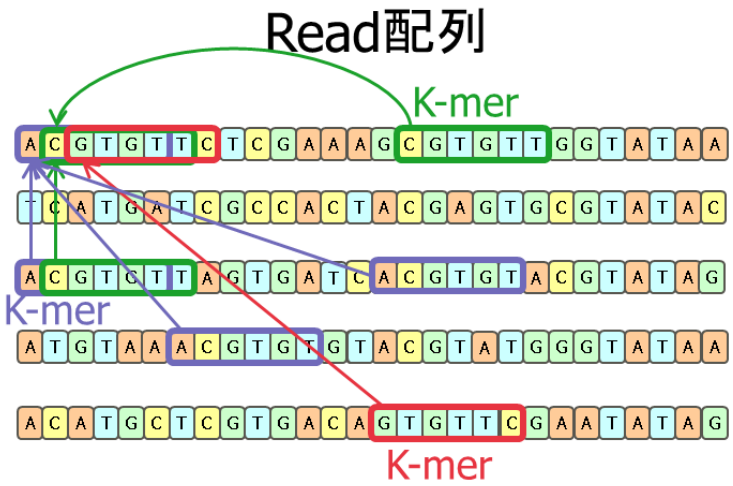

図 2 重複する K-mer がどこに出現するかを指すテーブル
Fig. 2 duplicated K-mer occurance table

を決めることである．その決め方は，様々なバリエーションが考えられるが，本稿では K-mer 値のハッシュ値の剰余によって決める．K-mer 値のハッシュ値の剰余値が自分が担当する剰余値のときだけ，K-mer インデックス処理を行う．自プロセスが担当する値以外の K-mer については処理を行わない．

図 3 は提案手法で，プロセスを 4 並列に分割した場合の処理を示している．この例で説明すると，K-mer 値”ACGTGT”が，ハッシュ値の剰余値が 0 の場合は，剰余値 0 を担当するプロセス 0 が，K-mer インデックス処理を行なう．同様に，K-mer 値のハッシュ値の剰余値が 1 の場合はプロセス 1，2 の場合はプロセス 2, 3 の場合はプロセス 3 が処理する．

各プロセスのハッシュテーブルには、自分が担当する K-mer のみ登録するが，K-mer インデックス処理は、同じ K-mer（これらは同じ剰余をもつ）をハッシュテーブルからみつける処理なので、すでに同じ K-mer が登録されていれば，必ず自プロセスのハッシュテーブルから見つけることができる．よって，自分が担当する K-mer についてだけ処理すれば，K-mer インデックス部分テーブル（自プロセスが担当する K-mer についての K-mer インデックステーブル）を作成することが可能である．

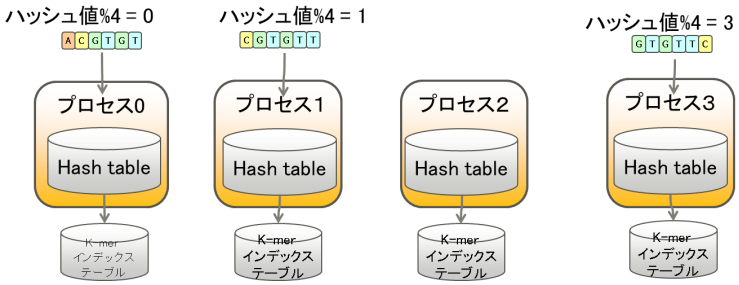

図 3 提案手法による velveth プロセス分割
Fig. 3 Process division of velveth

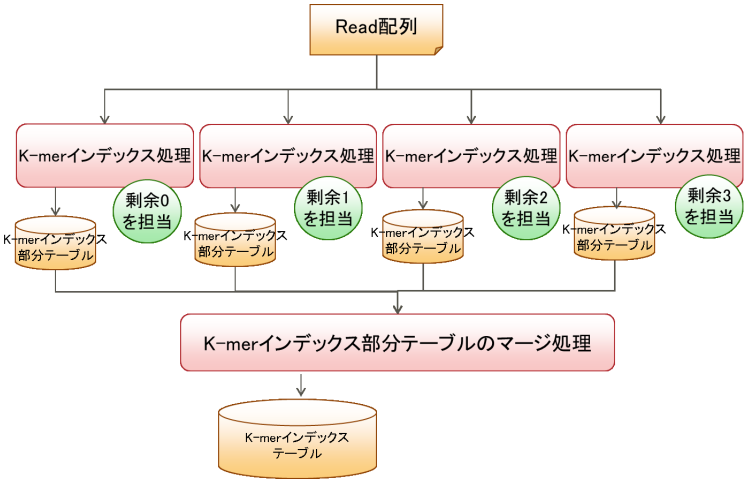

図 4 プロセス並列化 velveth の概要
Fig. 4 outline of process parallelized velveth

そして，各プロセスは自プロセスが担当する K-mer だけハッシュテーブルに登録するので，並列数が大きくなるほど，プロセスあたりのハッシュテーブルサイズが減少する．よって，この方式でプロセス並列化することで，プロセスあたり必要なハッシュテーブルメモリサイズを削減できる．

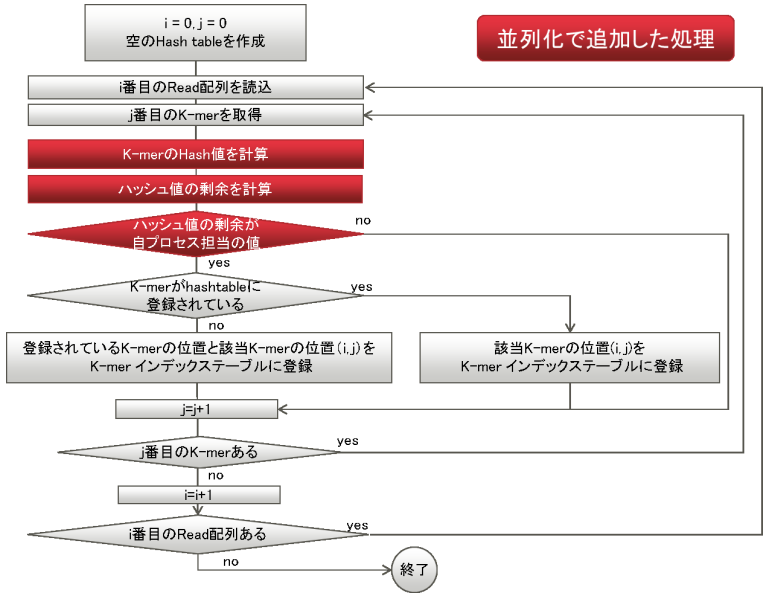

図 5 K-mer インデックス処理フローチャート
Fig. 5 flow chart of K-mer indexing procedure

4. 実　　装

4.1 K-mer インデックス処理

我々は，3 章で提案した手法を元にプロセス並列化 K-mer インデックス処理を実装した．プロセス並列化 velveth の処理概要を図 4 に，図 4 に対応するフローチャート図を図 5 に示した．本稿では，Velvet オリジナル版からの修正量がなるべく少なくなるように，Velvet オリジナル版の処理の流れを崩さない実装を選択した．実装方式によっては，ひとつのプロセスが Read 配列を読んで並列化プロセスに K-mer を分配するなどの構成も考えられるが，本稿の方式のほうが，オリジナル Velvet のバージョンアップへの追従が容易になる．

5. 評　　価

本章では，4 章で述べた方式で実装したプロセス並列化 velveth 実行時のメモリ消費量を測定した結果を示し，本稿で提案したプロセス並列化手法の有効性を評価する．

表 1 計算機環境
Table 1 Compute environment

Model	PRIMERGY BX960 S1
CPU	4 x Intel(R) Xeon(R) CPU L7555 (1.87GHz, 8-core)
Memory	128GB
Network	Giga Ethernet
File System	NFS

表 2 実行条件
Table 2 Running conditions

Software	Velvet 1.1.04
velveth options	K-mer length 31, -shortPaired, -fastq
Reads	Miscanthus giganteus (76bp paired x 23,280,317, SRR072869)

5.1 環　　境

本節では，評価に用いた評価環境について説明する．評価のために，**表 1**，**表 2** に記載した計算機環境と Read 配列を用意した．Read 配列は，逐次実行時のメモリ消費量も把握できるようにするため，評価用システムの搭載メモリ 128GB を超えないサイズのデータを用意して評価を行った．実際に研究室で利用する次世代シーケンサ Read 配列のサイズはより大きいと考えられる．

5.2 結　　果

提案手法を用いて実装した velveth 処理の，プロセス並列数 1, 2, 4, 8 で実行したときのプロセスあたりメモリ使用量 (全プロセスの最大値) を図 6 に記載する．逐次実行の場合では実行に 50.2GB のメモリを必要とした．プロセスを分割すると，2 分割で 26.3GB，4 分割で 14.2GB，8 分割で 7.8GB で，逐次実行時の消費量に比較して，それぞれ，約 1/2，約 1/3〜1/4，約 1/6 となった．

マージ処理が消費するメモリ量については，4 プロセス実行時に 4 ファイルをマージした時の値及び，8 プロセス実行時に 8 ファイルをマージした時の値が両方とも約 0.39GB であり，velveth 本体が消費するメモリ量 (14.2GB，7.8GB) と比較すると小さく，マージプロセスでメモリネックになることはなかった．

5.3 考　　察

前節の結果から，プロセス分割してプロセスあたりのメモリ消費量を減らせることがわかったが，本節でプロセス分割していくことでどこまで減らせるかを考察する．プロセスあたりのメモリ消費量はプロセス並列数の逆数に比例すると思われるので，プロセスあたりのメモリ量とプロセス並列数の逆数の関係が比例すると仮定して線形回帰分析を行った．その結果が式 (1) である．

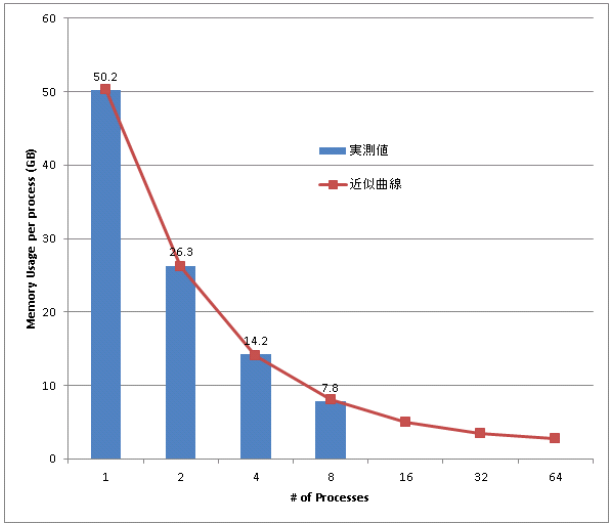

図 6 プロセス並列化 velveth プロセスあたりメモリ使用量
Fig. 6 Memory usage of a parallelized velveth process

$$M = -48.3/N + 1.98 \quad (1)$$

ここで，M はプロセスあたりのメモリ使用量（GB），N はプロセス数である．この回帰分析の決定係数は $R^2 = 0.9999$ で 1 に極めて近い値になっており，線形回帰式モデルの信頼度が高いことを表している．これによりプロセス並列数に反比例してプロセスあたりメモリ使用量が減少することが確認できる．切片が 1.98 であることから，プロセス並列数を増やしていくと，プロセスあたりメモリ使用量は 1.98GB に近づいていくことがわかる．このメモリ消費は，velveth が最初読み込む全 Read 配列のデータを固定的に保持していることが原因である．これは，各 Read 配列を読み込む度にその Read 配列についての k-mer インデックス処理を実施し，その度にメモリ領域を解放するように処理を改修することで，少なくなる見込みである．この部分のメモリ削減については，今後実装し評価を実施する予定である．

本稿では，提案プロセス並列化手法で実装した velveth 並列プログラムを評価を行い，プロセス並列数が多くなるほどプロセスあたりのメモリ消費量が少なくなることを確認した．以上から，velveth を提案手法で並列化することで，1 ノードではメモリ不足で実行できない Read 配列を処理する場合でも，PC クラスタのような分散メモリ環境を用意することで実行可能なことを確認できた．

6. おわりに

6.1 まとめ

本稿では，Velvet が大量にメモリを消費すること，特に Velvet を構成する velveth/velvetg プログラムのうち velveth のメモリ消費量が多いことを明らかにした．そこで，velveth の 1 ノードあたり使用メモリ量を削減し，分散メモリ環境で実行可能する velveth プロセス並列化手法を提案した．

提案手法では，velveth の K-mer インデックステーブル作成処理を，並列プロセスで実行する．そして，並列プロセスの各プロセスの担当 K-mer は，K-mer 値のハッシュ値を並列数で割った剰余によって決める．

この提案手法に従ってプロセス並列化 velveth を実装，評価を行い，逐次と比較して，プロセス 4 並列で 1/3 以下，プロセス 8 並列で 1/6 以下のメモリ使用量で各並列プロセスを実行可能であることを確認し，1 ノードではメモリ不足となって処理できない大規模な Read 配列でも，分散メモリ環境上のプロセス並列化 velveth で実行可能なることを確認した．

6.2 今後の課題

今後の課題としては，まずは 5.3 節で述べたように，メモリ消費削減可能な部分を効率化し，velveth 並列プログラムをさらにメモリ削減することである．また，本稿では着目しなかったが，velvetg プログラムの並列化にも着手しようと考えている．2 章で，velveth のほうが velvetg よりもメモリ消費量が多いと述べたが，velvetg のメモリ消費量も，Read 配列のサイズが大きくなるに従って増加し，大規模 Read 配列の処理でメモリ不足になることは velveth と同様である．よってプロセス並列化 velvetg を実現することで，Velvet 全体の処理で大規模 Read 配列のアセンブル処理を分散メモリ環境で実行可能にすることを目標に研究を行う予定である．

参考文献

1) Batzoglou, S. and et al.: ARACHNE: a whole-genome shotgun assembler., *Genome Res.*, No.12, pp.177–89 (2002).
2) Boisvert, S., Laviolette, F. and Corbeil, J.: Ray: Simultaneous Assembly of Reads from a Mix of High-Throughput Sequencing Technologies, *Journal of Computational Biology.*, Vol.17, No.11, pp.1519–1533 (2010).
3) Butler, J., MacCallum, I., Kleber, M. and et al.: ALLPATHS: De novo assembly of whole-genome shotgun microreads, *Genome Res.*, Vol.18, pp.810–820 (2008).
4) Haimei, M.: Velvet/Curtain, EMBL-EBI (online), available from ⟨http://www.ebi.ac.uk/training/ftp/private/onsite/NGS/NGS_EBI_presentation.pdf⟩ (accessed 2011-11-02).
5) Huang, X. and Yang, S.: Generating a genome assembly with PCAP, *Curr Protoc Bioinformatics Capter*, Vol.11(Unit11), p.3 (2005).
6) Illumina, inc.: De Novo Assembly Using Illumina Reads, illumina, Inc. (online), available from ⟨http://www.illumina.com/Documents/products/technote_denovo_assembly-ecoli.pdf⟩ (accessed 2012-02-23).
7) Miller, J.R., Koren, S. and Sutton, G.: Assembly Algorithms for Next-Generation Sequencing Data., *Genomics.*, Vol.95, No.6, pp.315–327.
8) Myers, E. and et al.: A whole-genome assembly of Drosophila., *Science*, Vol.287, pp.2196–204 (2000).
9) 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.
10) Ruiqiang, L. and et al.: De novo assembly of human genomes with massively parallel short read sequencing, *Genome Res.*, Vol.20, pp.265–272 (2010).
11) Simpson, J.T., Wong, K., Jackman, S.D. and et al.: ABySS: A parallel assembler for short read sequence data, *Genome Res.*, Vol.19, pp.1117–1123 (2009).
12) Zerbino, D.R. and Birney, E.: Velvet: Algorithms for de novo short read assembly using de Bruijn graphs., *Genome Res.*, Vol.208, No.18, pp.821–829.