Biochemical and Biophysical Research Communications 426 (2012) 395–398

Contents lists available at SciVerse ScienceDirect

Biochemical and Biophysical Research Communications

journal homepage: www.elsevier.com/locate/ybbrc

An efficient algorithm for DNA fragment assembly in MapReduce

Baomin Xu [a,*], Jin Gao [a], Chunyan Li [b]

[a] School of Computer and Information Technology, Beijing Jiaotong University, Beijing 100044, China
[b] Department of Computer Science, Tangshan Normal University, Tangshan 063000, China

ARTICLE INFO

Article history:
Received 10 August 2012
Available online 29 August 2012

Keywords:
Eulerian superpath problem
Parallel assembly
Fragment assembly
MapReduce

ABSTRACT

Fragment assembly is one of the most important problems of sequence assembly. Algorithms for DNA fragment assembly using de Bruijn graph have been widely used. These algorithms require a large amount of memory and running time to build the de Bruijn graph. Another drawback of the conventional de Bruijn approach is the loss of information. To overcome these shortcomings, this paper proposes a parallel strategy to construct de Bruijin graph. Its main characteristic is to avoid the division of de Bruijin graph. A novel fragment assembly algorithm based on our parallel strategy is implemented in the MapReduce framework. The experimental results show that the parallel strategy can effectively improve the computational efficiency and remove the memory limitations of the assembly algorithm based on Euler superpath. This paper provides a useful attempt to the assembly of large-scale genome sequence using Cloud Computing.

1. Introduction

A DNA sequence contains the genetic information of an organism. Obtaining the complete sequence of DNA and understanding the structure and composition of gene can greatly help grasp the organism's information. The sequence assembly technology is the basis of DNA sequence data processing. It can be used directly to determine the computational accuracy of gene identification, predicate the structure of protein, and perform many other gene-related functions. The sequence assembly techniques can be divided into two categories [21]: assembly with reference sequence, and assembly without reference sequence. In the former case, the assembly task is usually converted to the problem of finding the differences between the reference sequence and the assembled sample files [1]. The software systems that are widely used to do the analysis include Atlas-SNP, (developed by the U.S. Baylor College), and SOA (Short Oligonucleotide Analysis Package) [2–4] (developed by Huada Genomics Institute). In the latter case, the overlap-layout-consensus technique [5] is usually used. The basic idea of this technique is to gradually build a long sequence fragment by overlapping among fragments. This technique requires that the length of the sequence must be over 100bps, and the overlap be more than 30bps. With the development of biological sequencing technology, the length of the fragment obtained by the new generation sequencing technology is relatively shorter. The shortcomings of Overlap-layout-consensus technology are becoming more obvious. The most severe drawback of this technique is that it cannot solve repeat problem [6], which may cause the missing of some repeat regions or making two unrelated DNA fragments spliced to together.

To deal with this problem, Pavel A. etc. have proposed the Euler superpath in 2001 [7,8] and continually improved this algorithm since then. The main idea is to construct a de Bruijin graph and then find a Euler superpath on top of it. This algorithm has been used in many assembly software systems, such as Velvet [9], AllPath [10], and so on.

However, those systems demand a large amount of memory to construct a de Bruijin graph. As a result, it is very challenging to implement the assembly for a massive genome sequence on SMP (Symmetric Multi-Processing) processor systems. To solve this problem, many researchers designed and implemented parallel DNA sequence assembly algorithms [6,11,12]. With the prevalence of MapReduce, a cloud-based parallel computing framework [23], the studies on the high-performance sequence assembly algorithm using cloud have been conducted [13–20]. For example, CloudBurst [17] is a parallel stitching algorithm, which uses seed-and-extend, based on the MapReduce computational model. In 2011, Schatz MC etc. conducted some research on sequence assembly using de Bruijin graph based on MapReduce [18]. The basic idea was to divide a large graph into sub-graphs, which can then be executed on a number of parallel processors. When constructing the de Bruijin graph using MapReduce, each step requires both the information of the current node and that of all the other nodes in the graph. However, the creation of a de Bruijin graph may cause the loss of graph information. Furthermore, the accuracy of the splicing results may be affected. Therefore, if the de Bruijin graph is not

* Corresponding author. Address: School of Computer and Information Technology, Beijing Jiaotong University, Beijing 100044, China
E-mail addresses: xubaomin@gmail.com, bmxu@bjtu.edu.cn (B. Xu).

divided during the parallelization phase, the splicing results can be more accurate.

To our best knowledge, the sequence assembly algorithms using overlap-layout-consensus and de Bruijin technology based on MapReduce have not yet been published. In this paper, we proposed a parallel strategy without de Bruijin graph division, and presented a fragment assembly algorithm based on this strategy and MapReduce. The experimental results showed that our algorithm provided a better genome assembly solution than [18].

2. Methods

The processing of sequence assembly using de Bruijin graph follows three steps: (1) cut reads into k-mer; (2) construct the de Bruijin graph; and (3) find the Euler superpath.

2.1. Cut reads into k-mer

A Read is a sequence of fragments obtained by a sequencer. It is the input to the stitching program. The Euler superpath sequence assembly algorithm does not directly process reads. Instead, it cuts them into k-mers. The length of a k-mer is between 20 and 25. In our experiments, we set the k-mer length to 20.

Firstly, we divide a read file into several parts. These parts are then fed into the Map function. The Map function cuts each read into k-mers, which are in the form of key-value pair, where the key is a constant, and the value is the k-mer. These k-mers are then sent to the reduce function. The reduce function adds a serial number to each value k-mer, and then outputs a new set of key-value pairs, where the key is the serial number, and the value is the k-mer. These numbered k-mers become the nodes in the de Bruiijin graph.

2.2. Construction of de Bruijn graph

To construct a de Bruijin graph, we need to find the adjacency relationships between nodes (a fragment sequence). All the nodes are stored in HDFS (Hadoop Distributed File System), to ensure that that every map tasks can access the information of all nodes. The construction process is as follows:

(1) All k-mers generated by read splits are stored in a SequenceFile file (e.g., kmerfile) according to their numbers. The data format is key-value pair. i.e.,<number,k-mer>. SequenceFile can be accessed by the map and reduce functions.
(2) The kmerfile is the input to the stitching program. Firstly, the kmerfile is divided into a number of parts, each is an input to a Map task. The Map function reads the file row by row, obtaining the k-mer and its corresponding number. At this time, the Map function could get all the nodes' information by accessing SequenceFile. By traversing all the nodes and comparing each node to all the other nodes, the adjacency information about the current k-mer can be established. The adjacency information is a string containing all the nodes linking to the current k-mer. The Map function will generate a set of intermediate key-value pairs <number, adjacent information>.

For example, given two nodes k1 and k2 and their lengths |k1| and |k2|, the method to find the relationship between them is: Firstly, check whether the last |k1|-1 bases of k1 is the same to the first |k2|-1 bases of k2. If they are the same, k2 is added into the back adjacent node list of k1, and k1 is added into the forward adjacent node list of k2. If they are not the same, no action is performed. Secondly, check whether the last |k2|-1 bases of k2 is the same as the first |k1|-1 bases of k1. If that is the case, k1 is added into the back adjacent node list of k2, and k2 is added into the forward adjacent node list of k1. Otherwise, no operation is needed.

(3) The reduce function collects the <key, value >pairs generated by the map function, merges all intermediate values, and then saves them to a file adjfile. The adjfile contains the relationships of all the nodes. In fact, the adjfile is equivalent to a de Bruijin graph, which is created when we get the adjfile file.

This method solves the division of de Bruijin graph in a parallel process using SequenceFile, which can be considered globally accessible, because all the map functions can be linked together to this file. Dividing a de Brujjin graph can then be reduced to the tasks of dividing the file data, thus can be executed in parallel. The information of all the nodes in the graph can still be obtained in each map task. Therefore, the graph in fact is not divided, meaning that the de Bruijin graph is complete.

2.3. Finding Euler superpath

The basic idea is to choose a starting node as the seed node, and then find out a path based on the neighbor list and the path compatibility rules. Each step when searching the path is based on the relationships between nodes, and the next step depends strictly on the previous step. Obviously, the process does not need to parallelize.

(1) splicing processes

The relationships between nodes have been built after constructing the de Bruijin graph. For any node in the de Bruiijin graph, if there are adjacent nodes, the detailed neighbor information, such as match length, import, and export, can be obtained from adjfile. In practice, we select a node as the seed in the graph, splice from this node back and forth, until no adjacent nodes in both directions are found. The splicing process is listed as follows.

```
String seq = getKmer(startID);
int preID = startID;
int nextID = started;
while(preID != −1 || nextID != −1){
  if(preID != −1){
     seq = getKmer(preID) + seq;
    preID = preHashtable.get(preID);
  }
  if(nextID != −1){
    seq = seq = getKmer(nextID);
    nextID = nextHashtable.get(nextID);
  }
}
```

(2) Repeat sequence analysis

A node could have multiple adjacent nodes, due to repeat sequence. This problem can be solved by path compatibility. Specifically, during back splicing, if the current node has more than one adjacent node with the same match length, the algorithm will examine these adjacent nodes. If the entrance of the adjacent nodes is different from that of the current node, the two paths are not compatible, and the splicing does not implement. Otherwise, the two paths are compatible, and this adjacent node is selected to splice. During forward splicing, if the current node has more than one adjacent node, and the match lengths are the same, the program will examine these adjacent nodes. If the entrance of adjacent node is different from that of the current node, the two paths are not compatible, and the splicing does not implement. If the entrance of adjacent node and that of the current node are the same, the two paths are compatible. This adjacent node should be selected to splice. For example, for backward splicing, the process of path compatibility analysis is as follows.

```
boolean nextAdjID = getNextAdjID(kmerID);
ArrayList prelist = getList(nextAdjID);
for(int i = 0; i < preList.size(); i++)
{
   if(kmerID == prelist.get(i))
   {
      return true;
   }
}
```

3. Results and discussion

3.1. Conclusions

We developed a sequence assembly parallel algorithm using the MapReduce framework, which avoided the use of de Bruijin graph division. This algorithm is an innovative solution for DNA sequence assembly and analysis. The following conclusions can be drawn from our study: (1) Avoid graph division which can cause adverse impact on splicing results; (2) Overcome memory limitations in large sequence assembly by distributing de Bruijn graph; (3) MapReduce is an ideal solution for sequence assembly algorithm.

3.2. Test data

The test data we used are from NCBI (National Center of Biotechnology Information). We choose Bradyrhizobium (FJ555236.1), Octodon degus (AJSA01000011.1), and Homo sapiens collagen (NM_004370.5). Their lengths are 1987 bp¸3365 bp, and 11787 bp, respectively. The data processing procedure is as follows: (1) search aim genome which has been sequenced from Nucleotide database; get the aim genome sequence as the foundation of results evaluation; implement BLAST search using the sequence; and obtain the EST (expressed sequence tag) groups which are homologous as reads; (2) implement the assembly algorithm to get contigs; (3) evaluate the assembly results by DNA Blast Alignment Method algorithm.

3.3. Correctness verification of splicing results

The Correctness of the splicing results is the key to determine the effectiveness of our algorithm. We used both our algorithm and BioLign [22] to stitch three sets of data. The splicing results are shown in Table 1, and graphed in Fig. 1.

3.4. Verification of path compatibility

By cutting the reads, the Euler superpath algorithm solves the sequence repeat problem to some extent. The k-mers as the de Bruijin graph nodes also produce repeats. This will cause errors when splicing. We solve this problem by using path compatibility. When a repeat path appears during splicing, regardless of the cause, our algorithm will always select the correct path for splicing using the path compatibility principle. To find out the impact of path compatibility on our algorithm, two parallel algorithms with and without the use of path compatibility principle have been

Table 1
Experimental results.

Sequence name	Running environment	Comparing results
Bradyrhizobium	BioLign	0.8081
	Parallel algorithm	0.9910
Octodon degus	BioLign	0.7342
	Parallel algorithm	0.9878
Homo sapiens	BioLign	0.9076
	Parallel algorithm	0.9997

As seen in Table 1, the results of our parallel assembly algorithm are better than the results of BioLign. The reason is that our algorithm uses the path compatibility principle to solve the repeat problem, which leads to higher accuracy in the final sequence. Fig. 1 also shows that our algorithm is significantly better than BioLign. This is because BioLign does no give special handling to repeat sequences, so the splicing results vary in different DNA sequences. On the other handle, our parallel algorithm uses path compatibility to solve the repeat problem, so the results are more stable.

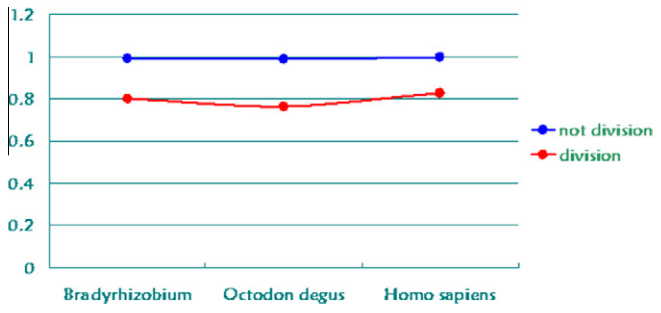

Fig. 1. Curve of experimental results.

Table 2
Experimental results.

Sequence name	Path compatibility	Comparing results
Bradyrhizobium	Not use	0.8563
	Use	0.9910
Octodon degus	Not use	0.7843
	Use	0.9878
Homo sapiens	Not use	0.8026
	Use	0.9997

It can be seen from Table 2 that the experimental results of the algorithm using path compatibility are better, and the splicing results have higher similarity to the original DNA sequence. It can be better seen from Fig. 2 that the splicing results of the two parallel algorithms have an obvious gap. The splicing results of the algorithm without the use of path compatibility have some fluctuations. The reason is that different DNA sequences have different repeats. In addition, the algorithm using path compatibility is not affected by DNA sequence. It can achieve better results regardless of the data set.

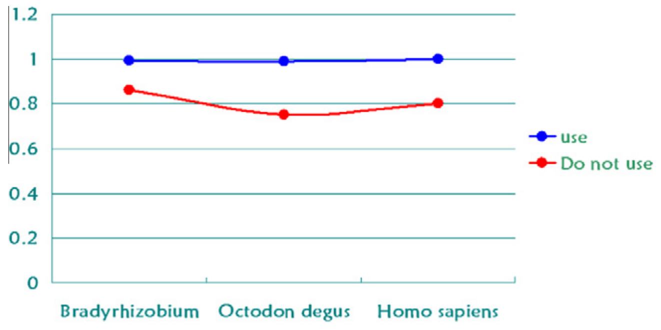

Fig. 2. Curve of experimental results.

Table 3
Experimental results.

Sequence name	Graph division	Comparing results
Bradyrhizobium	Without division	0.9910
	With division	0.8015
Octodon degus	Without division	0.9878
	With division	0.7632
Homo sapiens	Without division	0.9997
	With division	0.8267

As seen from Table 3, most of the splicing results of the algorithm without graph division have one to two percent higher similarity to the original DNA sequence than those of the algorithm with graph division reduces. Some splicing results' similarity is even lower than 80%. Fig. 3 also shows a clear gap between the splicing results of the two algorithms. These differences are mainly due to graph division, (a complete de Bruijin graph cannot be constructed, and some relationships between the associated nodes are lost). In addition, the splicing results of the two algorithms are stable, which proves the correctness of the proposed assembly algorithm.

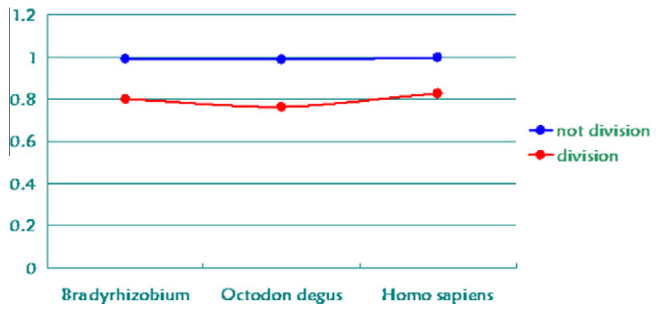

Fig. 3. Curve of experimental results.

implemented. The experimental results are shown in Table 2, the graph is shown in Fig. 2.

3.5. Algorithm verification

The sequence assembly algorithm based on de Bruijin graph division may cause some loss of graph information, and even some errors in the splicing results. This paper proposes a sequence assembly algorithm that does not use de Bruijin graph division. To investigate the impact of graph partitioning, we designed two parallel algorithms, one with and the other without graph division. The comparison results are shown in Table 3 and Fig. 3.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (NSFC60973011), Scientific Research Foundation of Doctoral Education (SRFDF20090009120007), and Jiangsu Provincial Natural Science Foundation of China (BK2011454).

References

[1] Michael Snyder, Du Jiang, Mark Gerstein, Personal genome sequencing: current approaches and challenges, Genes & Development 24 (5) (2010) 423–431.
[2] Ruiqiang Li, Yingrui Li, Xiaodong Fang, Huanming Yang, Jian. Wang, Karsten Kristiansen, Jun Wang, SNP detection for massively parallel whole-genome resequencing, Genome Research 19 (6) (2009) 1124–1132.
[3] Ruiqiang Li, Chang Yu1, Yingrui Li, Tak-Wah Lam, Siu-Ming Yiu, Karsten Kristiansen, Jun Wang, SOAP2: an improved ultrafast tool for short read alignment, Bioinformatics 25 (15) (2009) 1966–1967.
[4] R. Li, Y. Li, K. Kristiansen, J. Wang, SOAP: short oligonucleotide alignment program, Bioinformatics 24 (5) (2008) 713–714.
[5] Mihai Pop, Genome assembly reborn: recent computational challenges, Briefings In Bionformatics 10 (4) (2009) 354–366.
[6] Zheng Welmin, Lin Jiao, Luo Shubhua, A parallel. DNA fragment assembly algorithm. Based on Eulerian superpath, Journal of Bioinformatics 5 (2004) 91–101.
[7] Pevzner A. Pevzner, Haixu Tang, Michael S. Waterman, An Eulerian path approach to DNA fragment assembly, Proceedings of the National Academy of Sciences of the United States of America 98 (17) (2001) 9748–9753.
[8] Mark J. Chaisson, Pavel A. Pevzner, Short read fragment assembly of bacterial genomes, Genome Research 18 (2) (2008) 324–330.
[9] Daniel R. Zerbino, Ewan Birney, Velvet: algorithms for de novo short read assembly using de Bruijn graphs, Genome Research 18 (5) (2008) 821–829.
[10] J. Butler, I. Maccallum, M. Kleber, I.A. Shlyakhter, M.K. Belmonte, E.S. Lander, C. Nusbaum, D.B. Jaffe, Allpaths: de novo assembly of whole-genome shotgun microreads, Genome Research 18 (5) (2008) 810–820.
[11] M. Ahmed, I. Ahmad, S.U. Khan, A comparative analysis of parallel computing approaches for genome assembly, Interdisciplinary Sciences: Computational Life Sciences 3 (1) (2011) 57–63.
[12] A. Kalyanaramana, S.J. Emrichb, P.S. Schnablec, S. Aluru, Assembling genomes on large-scale parallel computers, Journal of Parallel and Distributed Computing 67 (12) (2007) 1240–1255.
[13] Ben Langmead, Kasper D. Hansen, Jeffrey T. Leek, Cloud-scale RNA-sequencing differential expression analysis with Myrna, Genome Biology 11 (8) (2010) R83.
[14] B. Langmead, M.C. Schatz, J. Lin, M. Pop, S.L. Salzberg, Searching for SNPs with cloud computing, Genome Biology 10 (11) (2009) R134.
[15] Massimo Gaggero, Simone Leo, Simone Manca, Federico Santoni, Omar Schiaratura, Gianluigi Zanetti, Parallelizing bioinformatics applications with MapReduce, in: Cloud Computing and its Applications, Chicago, Illinois, 2008.
[16] Michael C. Schatz, Ben Langmead, Steven L Salzberg, Cloud computing and the DNA data race, Nature Biotechnology 28 (7) (2010) 691–693.
[17] Michael C. Schatz, Cloudburst: highly sensitive read mapping with MapReduce, Bioinformatics 25 (1) (2009) 1363–1369.
[18] Schatz MC, Sommer D, Kelley D, Pop M. Assembly of Large Genomes with Cloud Computing, Illumina Sequencing Panel. Toronto, Canada, July 23, 2010.
[19] A. McKenna, M. Hanna, E. Banks, et al., The genome analysis toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data, Genome Research 20 (9) (2010) 1297–1303.
[20] Krithika Arumugam, Yu Shyang Tan, Bu Sung Lee, Rajaraman Kanagasabai, Cloud-enabling Sequence Alignment with Hadoop MapReduce, A Performance Analysis, 4th International Conference on Bioinformatics and Biomedical Technology, Singapore, 2012.
[21] Zhenyu Li, Yanxiang Chen, Mu Desheng, Jianying Yuan, Yujian Shi, Hao Zhang, Jun Gan, Nan Li, Hu Xuesong, Binghang Liu, Bicheng Yang, Wei Fan, Comparison of the two major classes of assembly algorithms: overlap–layout–consensus and de-bruijn-graph, Briefings in Functional Genomics 11 (1) (2012) 25–37.
[22] en.bio-soft.net/dna/BioLign.html.
[23] Xu Baomin, Chunyan Zhao, Hu Enzhao, Hu Bin, Job scheduling algorithm based on berger model in cloud environment, Advances in Engineering Software 42 (7) (2011) 419–425.