Small World Asynchronous Parallel Model for Genome Assembly

Jintao Meng[1,2,4], Jianrui Yuan[2,3], Jiefeng Cheng[2], Yanjie Wei[2*], and Shengzhong Feng[2*]

[1] Institute of Computing Technology, CAS, Beijing, 100190, PR.China,
[2] * Shenzhen Institutes of Advanced Technology, CAS, Shenzhen, 518055, PR. China
[3] Central South University, Changsha, 410083, PR. China
[4] Graduate University of Chinese Academy of Sciences, Beijing, 100049, PR. China
{jt.meng, jr.yuan, jf.cheng, yj.wei, sz.feng}@siat.ac.cn

Abstract. Large de bruijn graph based algorithm is widely used in genome assembly and metagenetic assembly. The scale of this kind of graphs - in some cases billions of vertices and edges - poses challenges to genome assembly problem. In this paper, a one-step bi-directed graph is used to abstract the problem of genome assembly. After that small world asynchronous parallel model (SWAP) is proposed to handle the edge merging operation predefined in the graph. SWAP aims at making use of the locality of computing and communication to explore parallelism for graph algorithm. Based on the above graph abstraction and SWAP model, An assembler is developed, and experiment results shows that a factor of 20 times speedup is achieved when the number of processors scales from 10 to 640 when testing on processing C.elegans data.

Keywords: parallel computing, De Bruijn graph, genome assembly

1 Introduction

Current sequencing technology (Illumina Solexa [1], Applied Biosystems SoLiD[2], and Helicos Biosciences Heliscope[3]) allows one to read millions of 35 to 100 nucleotide sequences per hour. Due to experimental errors, gaps, and genomic repeats, a much higher coverage depth of 50-fold to 300-fold is needed for accurate assembly. These factors contribute to a 300-fold to 1000-fold increase in the number of reads, resulting in billions of reads need to be processed, and this significantly complicates the genome assembly problem.

De Bruijn Assembler based on de-bruijn graph strategy [4][5] is well suitable for the current generation high throughput short reads assembly. In De Bruijn graph each vertex represents a length-k substring (k-mer) in a length-L read or its reverse complement. A directed edge connects two vertices u and v, if the $k-1$ length suffix of u is the same as the $k-1$ length prefix of v. Each input read is a path in the graph. By connecting such vertex pairs through edges, this approach generates the longest path without any branches as contigs.

The first De Bruijn assembler, EULER assembler [5], is proposed by Pevzner, who has transformed the fragment assembly problem into a variation of the classical Eulerian path problem by dividing reads into k-mers and then constructing k-mers into a path graph. This opens new possibilities on repeat resolution and generating error-free solutions for large-scale fragment assembly problem. Programs such as Velvet[6], SOAPdenovo[7], and IDBA[8] implicitly use this framework, but they are slightly different in details. Velvet handles these De Bruijn graphs efficiently by eliminating errors and resolving repeats using error correction algorithm. SOAPdenovo implements pre-assembler error correction on human genome assembly, after this operation the proportion of error free reads was improved from 64% to 70%, and nearly 60% percent of k-mers was filtered from the graph. IDBA also adopt pre-assembler error filtering technique, which can save nearly $40-80\%$ of memory compared with velvet. The second feature of IDBA is that it iterates from small k-mer to large k-mer to get longer contigs. So the quality of contigs is better than other tools.

The above assemblers can only run on single machine. Human genome assembly with current sequencing technology needs about 512GB memory and takes weeks or even months on single server. The performance is worse on plant genome assembly or meta-genome assembly.

Parallel algorithm for sequencing assembly is an alternative to solve the problem. Existing parallel assemblers, such as ABySS[9] and YAGA[10-12], are based on De Bruijn graph strategy. ABySS distributes k-mers to multi-servers to build a distributed De Bruijn graph, and error removal and vertex merging were implemented over MPI communication messages. YAGA constructs the distributed bi-directed De Bruijn graph by maintaining edge tuples in a community of servers. Unanimous chain compaction problem in YAGA was transformed to undirected list ranking, and then the authors designed a modified sparse ruling set algorithm for undirected lists. The computational complexity of YAGA is given by $O(\frac{n}{p})$ compute time, $O(\frac{n}{p})$ communication volume, and $O(log^2(n))$ communication rounds, where n is the number of nucleotides in all reads, and p denotes the number of processors.

Efficient and scalable frameworks or libraries for distributed graphs are essential to parallel assembly algorithms. Existing works, such as BSPlib [13-15], CGMgraph [16], PBGL [17,18], Prejel [19], are based on BSP [20] model. BSP model has advantage on simple computation-communication programming model, whereas barrel principle exists in the computation-communication phase and synchronous phase over large cluster limits the scalability of this model. The scalability of these implementations under BSP model has not been evaluated beyond several hundreds of computers [19]. No genome assembly tools have adapted these BSP library except that YAGA has used the BSP idea in its design on parallel list ranking algorithm. Another parallel programming model, MapReduce [21], has strength in loosely coupled work such as frequency statistics, sorting, indexing, and machine learning etc. However graph algorithm is a tight coupled work, and dividing one graph into several meaningful sub-graphs is still a challenging problem. Contrail [22] trys to transform the De Bruijn assembly problem to a list sorting problem, and it is the only parallel assembly tool based on Hadoop MapReduce platform. However the scalability and the quality of the output contigs are not further discussed.

This paper first demonstrates a one-step bi-directed graph for the problem of genome assembly. Genome can be recovered by by merging semi-extended edges to full-extended edges or contigs. Then small world asynchronous parallel (SWAP) model is proposed to realize edge merging over a distributed one-step bi-directed graph. Specially, we implement an assembler using the SWAP model. Given the number of processes p, the complexity of this problem is reduced to $O(\frac{n}{p})$ parallel compute time, $O(\frac{n}{p})$ communication round, and $O(\frac{glog(g)}{p})$ communication volume, here g is the length of genomes, and n is the number of nucleotide in all input reads. Simulation shows that this assembler over SWAP has a factor of 20 times speedup when the number of processors scales from 10 to 640.

The rest of the paper is organized as follows: Section 2 abstracts the De Bruijn graph based genome assembly problem; Section 3 describes the SWAP model for large scale graphs with small world property, then an assembler, as SWAP's first application, is illustrated. Experimental results will be present in section 4. Finally section 5 concludes this paper.

2 Abstraction of De Bruijn Assembly

DNA sequence is consisted of nucleotides coming from $\mathbb{N} = \{a, t, c, g\}$. Let $s \in \mathbb{N}^l$ be a DNA sequence of length l. Any substring derived from s with length k, denoted by $\alpha = s[j]s[j+1]\dots s[j+k-1], 0 \leqslant j \leqslant l-k+1$, is called a k-mer of s. The set of k-mers of a given string s can be written as $\mathbb{Z}(s, k)$, here k must be odd. The reverse complement of a k-mer α, denoted by α', is obtained by reversing α and complementing each base $a[i] = a[k-i+1]$ by the following bijection of $\mathbb{M}$, $\mathbb{M} : \{a \to t, t \to a, c \to g, g \to c\}$. Note that $a[i] = a[i]''$.

A k-molecule $\hat{\alpha}$ is a pair of complementary k-mers $\{\alpha, \alpha'\}$. Let $\gg$ be the partial ordering relation between the string of equal length such that $\alpha \gg \beta$ indicates that the string α is lexicographically larger than β. We designate the lexicographically larger of the two complementary k-mers as the positive k-mer, denoted as α^+, and the smaller one as the negative k-mer, denoted as α^-, here $\alpha^+ \gg \alpha^-$. We choose the positive k-mer α^+ as the representative k-mer of k-molecule $\{\alpha, \alpha'\}$, denoted as α^+. That means $\hat{\alpha} = \alpha^+ = \{\alpha^+, \alpha^-\} = \{\alpha, \alpha'\}$. The set of all k-molecules of a given string s is called the k-spectrum of s and is written as $\mathbb{S}(s, k)$. Noted that $\mathbb{S}(s, k) = \mathbb{S}(s', k)$.

We use the notation $suf(a, l)(pre(a, l)$, respectively) to denote the length l suffix (prefix, respectively) of string a. Let the symbol $\circ$ denotes the concatenation operation between two strings, and the number of edges attached to k-molecule $\hat{\alpha}$ is denoted as $degree(\hat{\alpha})$.

Definition 1. The vertex set V_s is defined as k-spectrum of s,

$$V_s = \mathbb{S}(s, k). \tag{1}$$

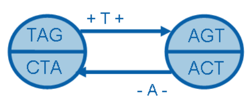
Definition 2. The 1-step bi-directed edge set E_s^1 is defined as follows:

$$\begin{aligned} E_s^1 = \{e_{\alpha\beta}^1 = (\alpha, \beta, d_\alpha, d_\beta, C_{\alpha\beta}^1) | \forall \hat{\alpha}, \hat{\beta} \in \mathbb{S}(s, k), suf(\alpha, k-1) = \\ pre(\beta, k-1) \wedge (\alpha \circ \beta[k-1] \in (\mathbb{Z}(s, k+1) \vee \mathbb{Z}(s', k+1)))\} \end{aligned} \tag{2}$$

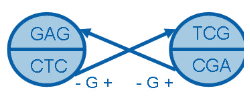
Equations (2) declares that any two overlapped k-molecules can be connected with a 1-step bi-directed edge, if they are continuous in sequence s or its complementary sequence. Here d_α denotes the direction of k-mer α, if $\alpha = \alpha^+$, $d_\alpha =' +'$, otherwise $d_\alpha =' -'$. $C^1_{\alpha\beta}$ is initialized with $\beta[k-1]$, and $suf(\alpha \circ C^1_{\alpha\beta}, k) = \beta$.

Property 1.Given two k-molecules $\hat{\alpha}, \hat{\beta} \in \mathbb{S}(s,k)$, there will be four possible connections, and for each type of connection exactly two equivalent 1-step bi-directed edge exist,

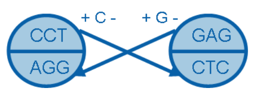
1. $e^1_{\alpha^+\beta^+} = (\alpha^+, \beta^+, +, +, C^1_{\alpha^+\beta^+})$, $e^1_{\beta^-\alpha^-} = (\beta^-, \alpha^-, -, -, C^1_{\beta^-\alpha^-})$
2. $e^1_{\alpha^+\beta^-} = (\alpha^+, \beta^-, +, -, C^1_{\alpha^+\beta^-})$, $e^1_{\beta^+\alpha^-} = (\beta^+, \alpha^-, +, -, C^1_{\beta^+\alpha^-})$
3. $e^1_{\alpha^-\beta^+} = (\alpha^-, \beta^+, -, +, C^1_{\alpha^-\beta^+})$, $e^1_{\beta^-\alpha^+} = (\beta^-, \alpha^+, -, +, C^1_{\beta^-\alpha^+})$
4. $e^1_{\alpha^-\beta^-} = (\alpha^-, \beta^-, -, -, C^1_{\alpha^-\beta^-})$, $e^1_{\beta^+\alpha^+} = (\beta^+, \alpha^+, +, +, C^1_{\beta^+\alpha^+})$



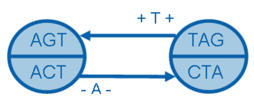
(a) Connection type 1



(b) Connection type 2



(c) Connection type 3



(d) Connection type 4

Fig. 1. The illustration of four possible connections.

In each type of connection, the first bi-directed edge and the second one correspond to the same bi-directed edge, but in different form. Whereas using this scheme, under a distributed edge representation, the first bi-directed edge in each type will be attached with k-molecule $\hat{\alpha}$, and the second one will be with $\hat{\beta}$. Figure (1) illustrates four possible connections and examples of a 1-step bi-directed edge graph.

Definition 3. 1-step bi-directed de bruijn graph of order k for a given string s can be presented as

$$G^1_k(s) = \{V_s, E^1_s\}. \tag{3}$$

Definition 4.Given two 1-step bi-directed edge $e^1_{\alpha\beta} = (\alpha, \beta, d_\alpha, d_\beta, C^1_{\alpha\beta})$ and $e^1_{\beta\gamma} = (\beta, \gamma, d_\beta, d_\gamma, C^1_{\beta\gamma})$, if $e^1_{\alpha\beta}.d_\beta = e^1_{\beta\gamma}.d_\beta$ and $degree(\hat{\beta}) = 2$, we can get 2-step bi-directed edge $e^2_{\alpha\gamma} = (\alpha, \gamma, d_\alpha, d_\gamma, C^2_{\alpha\gamma})$ by merging $e^1_{\alpha\beta}$ and $e^1_{\beta\gamma}$, here $C^2_{\alpha\gamma} = C^1_{\alpha\beta} \circ C^1_{\beta\gamma}$. Let the symbol $\bigoplus$ denote **edge merging operation** between two bi-directed edges attached to one k-molecule, then **edge merging operation** can be written as,

$$e^1_{\alpha\beta} \bigoplus e^1_{\beta\gamma} = e^2_{\alpha\gamma}. \tag{4}$$

or

$$e^1_{\gamma\beta} \bigoplus e^1_{\beta\alpha} = e^2_{\gamma\alpha}. \tag{5}$$

According to property 1, equation 4 and equation 5 correspond to one edge merging operation. Then z-step bi-directed edge can be defined as:

$$e^{z}_{\alpha\gamma} = e^{x}_{\alpha\beta} \bigoplus e^{y}_{\beta\gamma}, \mathit{iff}\ \exists\beta, z = x + y, e^{x}_{\alpha\beta}.d_\beta = e^{y}_{\beta\gamma}.d_\beta, degree(\hat{\beta}) = 2 \qquad (6)$$

Definition 5. Given an n-step bi-directed edge $e^{m}_{\alpha\beta} = (\alpha, \beta, d_\alpha, d_\beta, C^{m}_{\alpha\beta})$, if k-molecule α or β has only one another bi-directed edge $e^{t}_{\gamma\alpha} = (\gamma, \alpha, d_\gamma, d_\alpha, C^{t}_{\gamma\alpha})$ or $e^{t}_{\beta\gamma} = (\beta, \gamma, d_\beta, d_\gamma, C^{t}_{\beta\gamma})$ respectively, then $e^{m}_{\alpha\beta}$ can be extended by $e^{t}_{\gamma\alpha}$ or $e^{t}_{\beta\gamma}$, we call this edge a semi-extended edge, and the corresponding k-molecule α or β semi-extended vertex. If $e^{m}_{\alpha\beta}$ can not be extended by any edge, we call this edge a full-extended edge, k-molecule α and β full-extended vertex.

Given a set of string or reads $S = \{s_1, s_2, \ldots, s_h\}$, a one step bi-directed De Bruijn graph of S with order of k is $G^{1}_{k}(S) = \{V_S, E^{1}_{S}\} = \{ \bigcup_{1\leqslant i\leqslant h} V_{s_i}, \bigcup_{1\leqslant i\leqslant h} E_{s_i}{}^{1}\}$. The key property of this bi-directed De Bruijn graph $G^{1}_{k}(S)$ is that each read can be recovered by traversing the corresponding path in either direction, concatenating (k-1)-molecule prefix of the first node and the edge labels on the path. As all input reads of assembler are derived from chromosomes, each chromosome can now be seen as a long path in this graph. However because of read errors, and repeats in the sequence, we can not expect to see continuity in sampling, and our goal is to recover the genome as a large set of contigs by merging semi-extended edges to full-extended edges.

3 Assembler over SWAP

Vertices in large scale real world graph (such as social network, web link graph, et) always have limited number of neighbors, little computing work, and constant number of edges randomly connected to other vertices, this phenomenon is denoted as small world property. For a given vertex, we include all its edges, neighbors and itself in its small world. Then any computing and communication work of a vertex can be done in its small world. As long as the work of a vertex on a graph does not interrupt others, we can run computational work of those vertices in parallel. This section will introduce our work on pursuing parallelism in the computation of bi-directed graph for genome assembly.

Inspirited from CSMA/CA in wireless networks [23], Small World Asynchronous Parallel model (SWAP) aims at improving parallelism on processing large scale graph problem with small world property. After having distributed graph over a network of processors, the main schedule of SWAP can be defined as a combination of following three steps:

1. **Lock** operation is applied to each vertex's small world, which includes itself and its neighbors.
2. **Computation** and **modification** will be performed in each vertex's small world.
3. **Unlock** operation will be triggered after computation step.

The basic schedule of SWAP is Lock-Computation-Unlock. Because of the locality of computing and communication in the small world, SWAP model utilizes local synchronization and global asynchronization mechanism to maximize underline parallelism for the graph algorithm.

An assembler over SWAP is the first application using SWAP model. In the following paragraphs we will describe its data structure on distributed de brujin graph, strategy on error removal, and the edge merging algorithm, respectively.

3.1 Parallel construction of distributed one-step bi-directed De Bruijn graph

For m sequences, $S = \{s_1, s_2, \ldots, s_m\}$, sampled from a genome of total length g, we aim to construct a one-step bidirected De Bruijn graph $G_k^1(S)$ with $O(g)$ vertexes and edges distributed among p processors such that each processor stores $O(\frac{g}{p})$ vertices.

Input sequences can be broken into overlapping k-molecules by sliding a window of length k along the input sequence. Each processor maintains a hash table to store k-molecules, and each k-molecule is represented as a base-4 number of its positive k-mer. Numerical values $\{0, 1, 2, 3\}$ are assigned to bases $\{A, C, G, T\}$. The location of a given k-molecule can be computed by taking mod of a large prime number and then taking mod of the number of processors. The large prime number is used to evenly distribute k-molecules to all processors.

A single k-molecule can have up to eight edges, and each of them corresponds to a possible one-base extension, $\{A, C, G, T\}$ in either direction. The adjacent k-molecule can be easily generated by adding the base extension in the edge set to the source k-molecule.

The construction of one-step bi-directed De Bruijn graph can be achieved in $O(\frac{g}{p})$ parallel compute time, $O(1)$ round of all-to-all communication, and $O(\frac{g}{p})$ parallel communication volume. Here n is the total number of nucleotides in the input sequences.

3.2 Error removal

Sequencing errors make the assembly problem more complex. To identify errors, we assume that the errors are random, and they are unlikely to occur twice in the same base. As each base in the genome is sampled on average as many times as the coverage number, the erroneous k-molecule will have lower frequency compared to the correct ones. According to this principle, we identify all k-molecules with low frequency as erroneous k-molecules, and delete all of them from our vertex set of the graph. The complexity of this step is $O(\frac{g}{p})$ parallel compute time.

3.3 Edge merging

One step bi-directed graph generated in the previous step will likely have many long chains, and each corresponds to a sequence that can be unambiguously assembled into a single contig. We will merge these chains into full-extend edges with edge merging operation defined in section 2.

A subset of k-molecules and their associated edges are stored in processor i, the set of semi-extended k-molecules stored in this processor is denoted as V_i. The edge merging operation will be applied to all semi-extended edges to form full-extended

Algorithm 1 Edge Merging Operation Algorithm

```
Iteration:
1: Element selection operation
   For each semi-extended $k$-molecule $\hat{\alpha}$ in $V_i$, if $\hat{\alpha}$'s two neighbors $\hat{\beta}$ and $\hat{\gamma}$ are connected by
   edge $e^u_{\beta\alpha}$ and $e^v_{\alpha\gamma}$ , then $e^u_{\beta\alpha}$ and $e^v_{\alpha\gamma}$ can be merged as $e^{u+v}_{\beta\gamma} = e^u_{\beta\alpha} \bigoplus e^v_{\alpha\gamma}$, $e^{u+v}_{\gamma'\beta'} = e^v_{\gamma'\alpha'} \bigoplus e^u_{\alpha'\beta'}$
2: Lock
   In order to merging $e^u_{\beta\alpha}$, $e^v_{\alpha\gamma}$, we need to send Lock messages to lock k-molecules $\hat{\alpha}$, and its
   neighbours $\hat{\beta}$, $\hat{\gamma}$.
3: Computing
   $e^u_{\beta\alpha}$, $e^v_{\alpha\gamma}$ and $e^v_{\gamma'\alpha'}$, $e^u_{\alpha'\beta'}$ will be merged into one edge $e^{u+v}_{\beta\gamma}$. That means the original two edges
   $e^u_{\beta\alpha}$, $e^v_{\alpha\gamma}$ will be deleted, and new edge $e^{u+v}_{\beta\gamma}$, $e^{u+v}_{\gamma'\beta'}$ between $\hat{\beta}$ and $\hat{\gamma}$ will be added.
4: Unlock
   k-molecule $\hat{\alpha}$ sends unlock messages to unlock $k$-molecule $\hat{\alpha}$ , $\hat{\beta}$ and $\hat{\gamma}$.
Output:
   Return Full-extended edges.
```

edges or contigs. Algorithm of edge merging operation on SWAP implementation is described in Algorithm 1.

As the bi-directed graph $G^1_k(S)$ is distributed over p processors, each processor will store a subset of semi-extended k-molecules V_i, and the average number of k-molecules in V_i is $O(\frac{g}{p})$. Then the expected computational complexity of each processor on edge merging is given by $O(\frac{g}{p})$ parallel compute time, $O(\frac{g}{p})$ communication round, and $O(\frac{g \log g}{p})$ communication volume.

4 Experimental Results

The assembler is written in C++ and MPI. The hardware and software architecture supporting this assembler is demonstrated in Fig 2. We use Dawning 5000 as high performance cluster, which has 40 16-core servers with 32GB memory. The distributed file system is lustre. All the components are interconnected with infiniband 20Gbit Router.

Perl scripts [24] are used to generate the following two theoretical datasets: 50x coverage of Yeast chromosomes containing 17 million reads, and 50x coverage of C.elegans chromosomes containing 141 million reads. The error rate is set to be 1%, and the length of reads ranges from $36bp$ to $50bp$. The primary goal of this experiment is to demonstrate the scalability of SWAP model on handling large-scale graphs using parallel system with distributed memory.

We first test the performance of this assembler on Yeast dataset. The runtime of assembler is displayed in figure 3, and the time is divided into three phases, Parallel File I/O, graph building, and edge merging. The first phase is the time spent on reading dataset from a distributed file system, the second phase is the time used to construct the one-step bi-directed graph over the cluster, and the last phase is the time cost on edge merging operations. The run time is dominated by the third phase, where hundreds of processors are sending messages to lock their neighbors, merging edges, deleting semi-extended nodes and edges, and unlocking their neighbors. Figure 3 shows that this phase

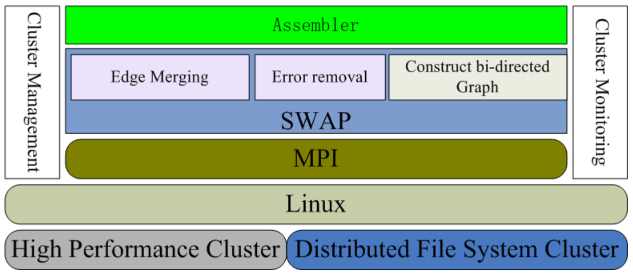


Fig. 2. Hardware and software architecture of the assembler over SWAP.

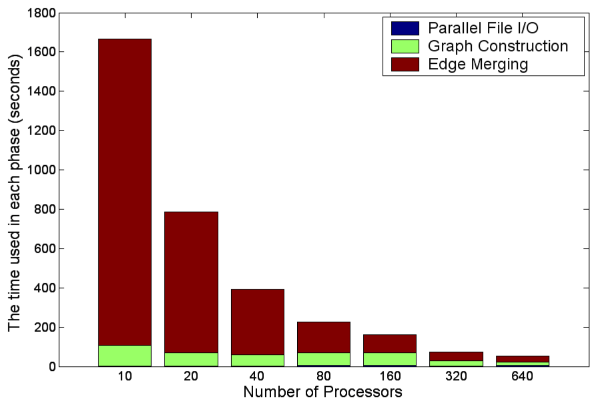


Fig. 3. Time usage analysis in three phase on Yeast dataset.

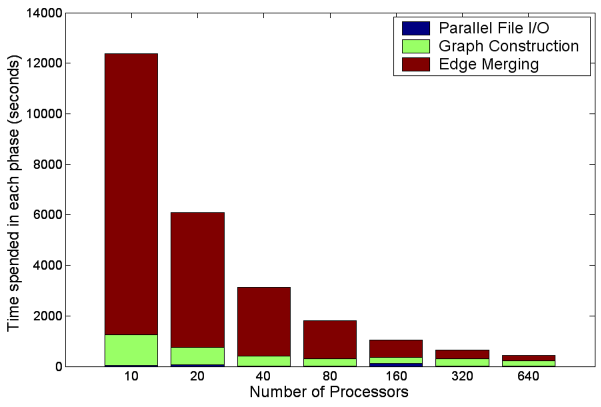


Fig. 4. Time usage analysis in three phase on C.elegans dataset.

has good scalability. The speedup is about 50 when the number of processor scales from 10 to 640 and the overall runtime of assembler on Yeast dataset is reduced by a factor of 30.

The C.elegans dataset is nearly ten times larger than Yeast, and its corresponding data on time usage is demonstrated in figure 4. In this figure, the time used in parallel file I/O is very short compared to the other two phases. The graph construction phase has a decreasing trend on its running time. This phase have a speedup of 35x. This speedup is slightly smaller than that of the yeast dataset. The total speedup of all three phases on C.elegans dataset is about 20.

5 Conclusion

In this paper, we abstracted the problem of genome assembly using De Bruijn strategy. By constructing the one-step bi-directed graph over k-spectrum of input sequences, the unanimous path compaction problem in generic genome assembly was transformed to merge semi-extended edges in our bi-directed graph, and the final contigs are full-extended edges in our method. The proposed SWAP introduced local synchronization and global asynchronization mechanism to maximize the parallelism in the graph algorithm. SWAP model applies the Lock-Computation-Unlock scheme to each vertex's small world. Based on SWAP model, we developed a De Bruijn assembler over SWAP, and simulation results confirm that when the number of processors scales from 10 to 640, a factor of 30 and 20 speedup, can be achieved on assembling Yeast and C.elegans genomes, respectively.

After taking edge merging operations on bi-directed graph, the size of this graph shrinks dramatically. The latter stage of genome assembly problem involves resolving branches with pair-end information can be done in a single machine. We will address this problem in our future work.

Acknowledgements

This work is supported by NSFC (Grant No. 61103049) and Shenzhen Research Fund (Grant No.JC201005270342A). The author also thanks Bingqiang Wang and his teammates from BGI, Prof. Francis Y. L. Chin, Siu-Ming Yiu from Hongkong University for their guidance and suggestions on this work.

References

1. S. Bennet., Solexa ltd.: Pharmacogenomics, vol.5, no.4, pp. 433–438, (2004)
2. V. Pandey, R.C. Nutter, E. Prediger: Applied Biosystems SOLiDTM System: Ligation-Based Sequencing. Next Generation Genome Sequencing: Towards Personalized Medicine (2008)
3. Business Wire: Helicos biosciences enters molecular diagnostics collaboration with renowned research center to sequence cancer-associated genes. Genetic Engineering and Biotechnology News (2008)
4. R.M. Idury, M.S. Waterman: A New Algorithm for DNA Sequence Assembly," Journal of Computational Biology. vol. 2, no. 2, pp. 291–306 (1995)

5. P.A. Pevzner, H. Tang, M.S. Waterman: An Eulerian path approach to DNA fragment assembly. In: Proceedings of the National Academy of Sciences of the United States of America (PNAS), vol. 98, no. 17, pp. 9748–9753 (2001)
6. D.R. Zerbino, E. Birney.: Velvet: algorithms for de novo short read assembly using De Bruijn graphs. Genome Research, vol. 18, no.5, pp. 821–829 (2008)
7. R. Li, H. Zhu, J. Ruan, W. Qian, X. Fang, Z. Shi, Y. Li, S. Li, G. Shan, K. Kristiansen, S. Li, H. Yang, J. Wang, J. Wang. De novo assembly of human genomes with massively parallel short read sequencing. Genome Research, vol. 20, no. 2, pp. 265–272 (2010)
8. Y. Peng, Henry C.M. Leung, S.M. Yiu, Francis Y. L. Chin: IDBA-A Practical Iterative De Bruijn Graph De Novo Assembler. In: RECOMB 2010, vol. 6044, pp. 426–440 (2010)
9. J.T. Simpson, K. Wong, S.D. Jackman, J.E. Schein, et al: ABySS: a parallel assembler for short read sequence data. Genome Research, vol. 19, no. 6, pp. 1117–1123 (2009)
10. B.G. Jackson, S. Aluru: Parallel Construction of Bidirected String Graphs for Genome Assembly. In: Proceeding of the 37th International Conference on Parallel Processing (ICPP'08), pp. 346–353 (2008)
11. B.G. Jackson, P.S. Schnable, S. Aluru: Parallel short sequence assembly of transcriptomes. BMC Bioinformatics (2009)
12. B.G. Jackson, M. Regennitter, X. Yang, P.S. Schnable, S. Aluru: Parallel de novo assembly of large genomes from high-throughput short reads. In: Proceeding of the 24th International Symposium on Parallel & Distributed Processing (IPDPS'10), Atlanta, pp. 1–10 (2010)
13. Richard Miller: A Library for Bulk-Synchronous Parallel Programming. In: Proceeding of British Computer Society Parallel Processing Specialist Group Workshop on General Purpose Parallel Computing (1993)
14. M.W. Goudreau, K. Lang, S.B. Rao, T. Suel, T. Tsantilas: Portable and Effcient Parallel Computing Using the BSP Model. IEEE Transactions on Computers, vol. 48, no. 7, pp.670–689 (1999)
15. O. Bonorden, Ben H.H. Juurlink, Ingo von Otte, I. Rieping: The Paderborn University BSP (PUB) Library. Parallel Computing, vol. 29, no. 2, pp. 187–207 (2003)
16. A. Chan, F. Dehne: CGMGRAPH/CGMLIB: Implementing and Testing CGM Graph Algorithms on PC Clusters and Shared Memory Machines. International Journal of High Performance Computing Applications, vol. 19, no. 1, pp. 81–97 (2005)
17. D. Gregor, A. Lumsdaine: The Parallel BGL: A Generic Library for Distributed Graph Computations. In: Proceeding of Parallel Object-Oriented Scientific Computing (POOSC), (2005)
18. D. Gregor, A. Lumsdaine: Lifting Sequential Graph Algorithms for Distributed-Memory Parallel Computation. In: Proc of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (OOPSLA'05) (2005)
19. G. Malewicz, M.H. Austern, Aart J. C. Bik, J.C. Dehnert, I. Horn, N. Leiser, G. Czajkowski: Pregel: a system for large-scale graph processing. In: SIGMOD'10 Proceedings of the 2010 international conference on Management of data, pp. 135–146, New York (2010)
20. Leslie G. Valiant: A bridging model for parallel computation. Communications of the ACM, vol. 33, no. 8 (1990)
21. J. Dean, S. Ghemawat: MapReduce: simplified data processing on large clusters. Communications of the ACM - 50th anniversary issue: 1958 - 2008, vol. 51, no. 1, USA (2008)
22. M. Schatz, D. Sommer, D. Kelley, M. Pop: Contrail: Assembly of Large Genomes using Cloud Computing, `http://schatzlab.cshl.edu/presentations/2010-07-23.Illumina.pdf`
23. Andrew S. Tanenbaum: Computer Networks, Prentice Hall, New Jersey (2003)
24. W. Zhang, J. Chen, Y. Yang, Y. Tang, J. Shang, B. Shen: A practical comparison of de novo genome assembly software tools for next-generation sequencing technologies. PLoS ONE, vol. 6, no. 3 (2011)