Performance analysis of parallel \textit{de novo} genome assembly in shared memory system

Syam Budi Iryanto$^{1,2}$, Wisnu Ananta Kusuma$^1$, Heru Sukoco$^1$

$^1$Department of Computer Science, School the Graduate Studies, Bogor Agricultural University, Bogor, Indonesia
$^2$Research Center for Biomaterials, Indonesian Institute of Sciences, Bogor, Indonesia

E-mail: syam.iryanto@apps.ipb.ac.id

Abstract. \textit{De novo} genome assembly is computationally intensive tasks in genome analysis, where it builds the whole genome from small fragments (reads) generated by next-generation sequencing (NGS) platform. Parallel processing is a method to reduce the time complexity. In this work, we analyze the performance of three popular \textit{de novo} genome assembly tool based on \textit{de Bruijn} graph i.e., Velvet, SOAPdenovo2, and ABySS in a parallel environment. Simulated and real genome datasets from several species are used in this study. We determine the performance using two criteria, including the quality of contigs produced and the parallel performance. For the quality of contigs produced, we measure the N50 size, the number of contigs, and maximum contigs length. As for the parallel performance, we measure the speedup of the use of multi-core CPU in a shared memory system. Lastly, memory usage for each tool also compared. Based on the experiment, SOAPdenovo2 have the best performance for the quality of contigs produced with highest N50 value. All assembly tool work well in the parallel environment and give the speedup significantly. SOAPdenovo2 is the best tool that gives 22 times super-linear speedup. As for memory usage, ABBySS is the most efficient one.

1. Introduction

Bioinformatics is a field of study that consists of multi-area of sciences, including biology, mathematics, and statistic sciences. A biological experiment such as DNA sequencing produced a massive of data that should be analyzed. DNA Sequencing is a process of determining the precise order of nucleotides. DNA sequencing technology has been evolved since the 1970s, started by Gilbert and Maxam [1], Sanger and Coulson [2], until the development of next-generation sequencing (NGS) several years ago. NGS technology has been revolutionized research in the field of biology and medicine. Several NGS platforms including Roche 454, Illumina Solexa, and ABI-SOLiD technologies work on the principle of sequencing millions of DNA fragments simultaneously in a massively parallel mode to produce sequence data in megabases or gigabases [3]. Besides that, the use of NGS also reduced the cost required to sequence than the previous generation sequencer. [4].

Shotgun method is the most used technology for genome sequencing. In this technique, genomes are randomly cut into many small fragments, and a computer application is required to reconstruct the fragments into long DNA sequence. To reconstruct the small fragments into a long sequence, we need to map the fragments into genome’s reference, or if the reference is not available, then we use \textit{de novo} assembly technique.
De novo DNA assembly techniques can be categorized into two approaches, i.e., overlap-layout-consensus (OLC) and Eulerian path using de Bruijn graph. Reads generated by NGS platform have a very short size. Therefore OLC approach is not suitable because it will make the overlapping stages more difficult [5]. The difficulty of OLC approach to handling very short reads has to motivate Pevzner [6] to introduce Eulerian path using de Bruijn graph. Several commonly-used assembly tools based on de Bruijn graph include Velvet [7], SOAPdenovo2 [8], and ABySS [9].

To evaluate the performance of DNA assembly tool, the most common metrics used including the N50 size, the number of contigs, and the maximum of contigs length. Due to continuously grow in size and the complexity of the sequenced organism, hardware resources performance also need to be evaluated, such as memory usage and the speedup of parallel computation.

Previous research has been conducted to evaluate the performance of de novo assembly tools. Lin et. al. [10] compare several assembly tools based on OLC and de Bruijn graph by using some metrics including N50 length, sequence coverage, and assembly accuracy. The research also compares the computational time and memory usage. Zang et. al. [11] compare eight tools based on OLC and de Bruijn graph using several criteria. The research indicates that string-based assemblers are well-suited for very short reads and OLC assemblers for longer reads of small genomes.

In this paper, we present the performance evaluation of three commonly-used de novo assembly tool based on de Bruijn graph including Velvet, SOAPdenovo2, and ABySS. We evaluate the performance of the tools using two criteria, i.e., the quality of contigs produced and the parallel performances. For the quality of contigs produced, we measure the N50 size, the number of contigs, and the maximum contigs length. As for parallel performance, the speedup and memory usage are measured. To examine the quality of contigs produced, a simulated dataset is used, while to determine the parallel performance, we use real dataset.

1.1. Materials and Methods
This section is divided into four subsections, i.e., dataset, de novo assembly computation, performance evaluation by measuring the quality of contigs produced, and parallel performance evaluation.

1.2. Dataset
This study uses two type datasets, simulated and real dataset. Simulated dataset generated using GemSIM simulator [12] with Illumina error models. GemSIM is NGS simulator capable of creating single/paired-end reads. The input data for GemSIM is a FASTA format from organism shown in Table 1, and the output data is in FASTQ format. With GemSIM, we can simulate data with custom read length and coverage depth. To evaluate the performance of parallel environment, we also use real dataset downloaded from NCBI database. We use middle and large size genome read archive as shown in Table 2.

| Organism           | Genomelength (bp) |
|--------------------|-------------------|
| E.coli             | 5.3 M             |
| A.xylosoxidans     | 6.9 M             |
| M.stipitatus       | 10.3 M            |
Table 2. Sequence archive (SRA) from NCBI for real dataset

| Organism      | Genome length (bp) |
|---------------|--------------------|
| ERR930305     | 888 M              |
| SRR001665     | 749.4 M            |
| SRR493234     | 8.1 G              |

Figure 1. Assembly process workflow

1.3. De novo assembly computation

In this work we use three commonly-used assembly tools based on de Bruijn graph and designed for NGS platform, i.e., Velvet, SOAPdenovo2, and ABYSS. Velvet is designed for very short reads, about 25-50 bp [7]. In this experiment, we use Velvet 1.1 which already support parallel computation with multi-threading scheme. SOAPdenovo2 is an assembly tool that designed to assembly human genome read archive and specially designed for Illumina NGS. SOAPdenovo2 is an improvement tool from their successor, SOAPdenovo, that is more efficient in memory usage. ABYSS is a tool that used distributed de Bruijn graph. Thus it can be work in a distributed system. ABYSS also designed for assembly the human genome. In this experiments, we only use in shared memory environment. The workflow of this experiments shown in Figure 1.

NGS Simulator firstly generates the sequence read archive from FASTA format dataset to produce FASTQ file format. The NGS simulator used to generate sequence read archive with custom result, e.g., variation of coverage depth, custom read length, etc. The output from NGS simulator become an input in assembly tool to produce contigs. To evaluate the performance of assembly tool, we use N50 statistic value. To evaluate the hardware resources performance, we calculate the speedup and measure the memory usage.

This research conducted on a compute node with dual-core Intel Xeon 2650 2GHz processors and 256GB RAM. The node connected to the master node with dual gigabit NIC. The operating system used in both nodes are Centos Linux 6.6 with Torque 4.2.6 as the resource management and MAUI 3.3.1 as a job scheduler. Master node and compute node are a part of cluster...
computer called Grid@LIPI, located in Research Center for Informatics, Indonesian Institute of Sciences (LIPI).

1.4. Performance evaluation by measuring the quality of contigs produced
Performance evaluation for simulated datasets are measured using N50 length, a statistical method defined as the largest length of \( C \) such that 50\% of all contigs length is contained in contigs of size at least \( C \) \[13\]. N50 length is a standard measure of assembly connectivity. The higher N50 length, the better assembly result.

1.5. Parallel performance evaluations
De novo assembly is a computationally intensive task, especially when it is used to assemble large sequence archive, e.g., the human genome. Significant memory allocation occurs when reads are indexed and extracted to \( k \)-mer. Memory usage reports are obtained from PBS servers sent via email to the user.

In addition to the use of memory resources, assembly computation also needs a reliable and fast CPU clock. To reduce the time complexity, parallel processing is required. Assembly tools used in this experiments has supported a parallel processing. In this study, parallel computation was performed to assemble the genome read archive for organisms from the real dataset. Evaluation performance measured by calculating the speedup and the efficiency using formula (1) and (2).

\[
S_p = \frac{T_1}{T_n} \quad (1)
\]

\[
E_p = \frac{S_p}{p} \quad (2)
\]

\( S_p \) is speedup, \( T_1 \) is time used for computation using 1 CPU core, \( T_n \) is time used for computation using \( n \) CPU cores. Ideal speedup occurs when \( S_p = p \), i.e., the speedup is equal to the number of CPU cores used. \( E_p \) is an efficiency value, measured by dividing the speedup with the number of CPU core. Ideal value for \( E_p \) is equal to 1, where \( S_p = p \).

2. Results and Discussions
2.1. The dataset generated by simulator
Simulated dataset generated by GemSIM using Illumina error models. GemSIM is running on Linux operating system using a command line and does not have a graphical interface. Input sequence used by GemSIM is a file with FASTA format. Example command for generating read archive sequence:

```
./GemReads.py -r myxococcus.fna \
-n 25876465 -l 36 -u d  \
-m models/ill100v5_p.gzip -q 64 \
-o myxococcus/myxococcus-illumina-cov90x -c
```

GemSIM output is a paired-end FASTQ file. In this work, we conducted the adjustment for read length equal to 36 bp and coverage depth varies, ranging from 10\( \times \) to 90\( \times \).

2.2. Performance of assembly tool by \( k \)-mer length, coverage depth, and N50
\( k \)-mer length \((k)\) adjustment is essential factor to get an optimal assembly result. In this step, we calculate the optimal value of \( k \) for each coverage depth. From the Table 3, we noticed that the higher the coverage depth, the higher the \( k \) value. The table also shows that when the coverage depth is 50\( \times \), the \( k \) value tends to be stable and does not rise significantly for the
higher coverage depth value. The value of $k$ is used when constructing de Bruijn graph, where it would determine the position of the read will be truncated. The $k$ value that is too small will make the resulting contigs too short and too many connections. On the other hand, if the value is too big, it will make long contigs but with a few connections.

To get an optimal coverage depth that makes N50 size higher, we calculate the N50 value with coverage depth ranging from 10× to 90× for all simulated datasets. The graphic in Figure 2, 3, and 4 show the results. For all simulated datasets, SOAPdenovo2 have the highest value of N50 compared to Velvet and ABySS. From the figures, N50 value tends to be stable when the coverage depth is 50× and does not rise significantly for the higher coverage depth value. Theoretically, a high coverage depth can overcome the error problem in read caused by NGS. With the high coverage depth, the same $k$-mer is increasingly emerging, and the edges in the graph that lead to the $k$-mer will have high frequency. $K$-mer that generated by error read, potentially become a branch in a graph and have low frequency. From the frequency information, the algorithm can decide whether the $k$-mer is an error or not.

### Table 3. Optimal $k$-mer length

| Assembly tool | Organism          | Coverage depth |
|---------------|-------------------|----------------|
|               | 10×               | 20×            | 30×            | 40× | 50× | 60× | 70× | 80× | 90× |
| Velvet        | A.xylosoxidans    | 23             | 27             | 29  | 29  | 29  | 29  | 31  | 31  |
|               | E.coli            | 21             | 25             | 27  | 29  | 31  | 31  | 31  | 31  |
|               | M.stipitatus      | 23             | 25             | 29  | 29  | 31  | 31  | 31  | 31  |
| ABysS         | A.xylosoxidans    | 23             | 27             | 29  | 31  | 33  | 33  | 33  | 33  |
|               | E.coli            | 21             | 25             | 29  | 31  | 31  | 33  | 33  | 33  |
|               | M.stipitatus      | 25             | 27             | 29  | 31  | 33  | 33  | 33  | 33  |
| SOAPdenovo2   | A.xylosoxidans    | 21             | 25             | 27  | 29  | 31  | 31  | 33  | 33  |
|               | E.coli            | 19             | 25             | 27  | 29  | 31  | 31  | 31  | 31  |
|               | M.stipitatus      | 19             | 25             | 27  | 31  | 31  | 31  | 31  | 31  |

### Figure 2. N50 A.Xylosoxidans.
Table 4. Output comparison of assembly tool with 80 coverage depth

| Organism     | Metrics         | Velvet | SOAPdenovo2 | ABySS |
|--------------|-----------------|--------|-------------|-------|
| A.xylosoxidans | Number of contigs | 1550   | 1917        | 2172  |
|              | N50             | 2531   | 33938       | 27863 |
|              | Max contigs length | 96311 | 102681      | 125580|
| E.coli       | Number of contigs | 1435   | 2372        | 2257  |
|              | N50             | 27795  | 31114       | 29884 |
|              | Max contigs length | 137103 | 147566      | 137107|
| M.stipitatus | Number of contigs | 3293   | 2347        | 3164  |
|              | N50             | 26340  | 34684       | 29963 |
|              | Max contigs length | 144396 | 103237      | 103241|

The quality of contigs produced by assembly tool evaluated using three parameters i.e., number of contigs, N50 size, and maximum contigs length. The expected results for contigs is a contigs with a minimal number, higher N50 score, and longer maximum contigs length. From the experiments, the highest N50 score for each dataset is by SOAPdenovo2. As shown in Table 4, the highest value of N50 size achieved by SOAPdenovo2 for all organisms. This results indicate that error checking algorithm in SOAPdenovo2 works well. SOAPdenovo2 implement error correction algorithm based on k-mer frequency spectrums (KFS), called SOAPec-2.0 [8].
2.3. Performance of assembly tool by memory usage

De novo assembly computation is a task that uses memory device very intensive. Read sequence archive is a file that contains a million or billion characters that must be allocated in a memory device. The bigger the size of reads, the bigger memory required. Each assembly tools have their algorithm to manage this file on a memory device. As shown in Figure 5, 6, 7, the higher coverage depth, memory required by Velvet is increased, but it does not occur in SOAPdenovo2 and ABySS. This information is essential when we have to choose assembly tool to assemble high coverage read archive, especially if our computing resources are limited, we can choose the right tool.

![Figure 5. Memory usage of A.Xylosoxidans.](image1)

![Figure 6. Memory usage of E.Coli.](image2)

![Figure 7. Memory usage of M.Stipitatus.](image3)

This work also investigate the memory usage for middle and large size read archives, including ERR930305, SRR001665, and SRR493234. The detail results for this dataset shown in Table 5. From the table, SOAPdenovo2 need bigger memory than Velvet and ABySS in SRR493234 and ERR930305. For all read archive, ABySS need less memory than Velvet and SOAPdenovo2. ABySS only require about 33% and 50% of SOAPdenovo2 required for large and middle size genome, respectively.
Table 5. Memory usage for real dataset

| Organism     | Genome length (bp) | Cov | Velvet | SOAPdenovo2 | ABySS |
|--------------|--------------------|-----|--------|-------------|-------|
| ERR930305    | 888.9 M            | 171 | 1.7    | 5.1         | 1.9   |
| SRR001665    | 749.4 M            | 163 | 3.1    | 2.3         | 1.9   |
| SRR493234    | 8.1 G              | 2.5 | 92.3   | 123.0       | 47.3  |

Figure 8. Speedup for SRR493234.

Figure 9. Efficiency for SRR493234.

2.4. Performance of assembly tool by speedup

To reduce the time complexity for de novo assembly computation, parallel computation is the best approach. All assembly tools used in this research has supported parallel computation. In this work, we investigated the use of multi-core processors by calculating the speedup. Figure 8 shows the speedup for all assembly tool when it is used to assemble SRR493234 read archive.

As shown in Figure 8, we noticed a super-linear speedup in SOAPdenovo2 when 4 and 8 CPU cores used. We speculate is due to SOAPdenovo2 works using parallel threads. From the graph, when 12-16 CPU cores used, the speedup tends to decrease if compared with the previous result. The decreased of speedup is caused by the communication cost needed between processors. The compute node used in this experiment has two processors with eight core each and connected by front side bus.

Sub-linear speedup occurs in Velvet and ABySS with efficiency value less than 1, as shown in Figure 9. We speculate due to both assembly tools does not optimize memory usage, as shown
in Table 3 although physical memory capacity is large enough, however memory used by Velvet and ABySS only 37% and 18% respectively from the total capacity.

3. Conclusions
In this work, we analyze the performance several popular assembly tools including Velvet, SOAPdenovo2, and ABySS using several criteria. We calculate the optimal $k$-mer length, coverage depth, and also N50 for each assembly tool. To evaluate the performance of the resources, we measure the use of memory resources and calculate the speedup when the tools used for assembly large genome archive sequences. Simulated dataset generated by GemSIM simulator and real sequence archive dataset downloaded from NCBI are used in this study.

To get a good assembly result, optimal $k$-mer length settings is required. From the experiment results with a simulated dataset, optimal $k$-mer length value for each coverage depth are different, the higher the coverage, the higher the value of optimal $k$-mer. The best N50 size for 36 bp read is 50 coverage depth with optimal $k$-mer length 31-33.

Memory required by Velvet is depended on the coverage depth, the higher the coverage depth, the more memory needed. This does not occur on SOAPdenovo2 and ABySS, where the required memory in both tools is the same for all coverage depth. In the case computation for large genome archive, compared to SOAPdenovo2 and Velvet, ABySS requires the least amount of memory. The use of multi-core CPUs produces super-linear speedup on SOAPdenovo2 and sub-linear speedup on Velvet and ABySS. Maximum speedup achieved by SOAPdenovo is 22 times, i.e., when using 12 core CPUs.

Although this research has successfully analyzed the quality of contigs produced by the assembly tools, however, this research does not verify the resulted contigs by aligning into genome reference. Future work will focus on another parallel environment such as using Graphical Processing Unit (GPU) to accelerate the computation.

4. Acknowledgement
This research was conducted on Grid®LIPI facilities, Research Center for Informatics, Indonesian Institute of Sciences (LIPI), Indonesia.

5. References
[1] Gilbert W and Maxam 1973 Proc. Natl. Acad. Sci. U.S.A. vol 70 p 3581
[2] Sanger F and Coulson 1975 J. Mol. Biol 94 441
[3] Gupta AK and Gupta UD 2014 Anim. Biotechnol. 19 345
[4] Wetterstrand 2015 Available at www.genome.gov/sequencingcosts Accessed [November 9, 2017]
[5] Kusuma et. al. 2011 IPSJ Transactions on Bioinformatics 4 21
[6] Pevzner et. al. 2001 Proc. Natl. Acad. Sci. vol 98 9478
[7] Zerbino DR and Birney E 2008 Genome Res 18 821
[8] Luo et. al. 2012 Gigascience 1 18
[9] Simpson et. al. 2009 Genom Res 19 1117
[10] Lin et. al. 2011 Bioinformatics 27 2031
[11] Zang et. al. 2011 PloS ONE 6 e17915
[12] Mcelroy et. al. 2012 BMC Genomics 13 74
[13] Lander et. al. 2001 Nature 409 860