EPiC Series in Computing

Volume 60, 2019, Pages 36–47

Proceedings of 11th International Conference on Bioinformatics and Computational Biology

DALIGNER Performance Evaluation on the Xeon Phi Architecture

Evaldo B. Costa, Gabriel P. Silva, and Marcello G. Teixeira

Computer Science Department
UFRJ
Rio de Janeiro, Brazil
evaldo.costa@ppgi.ufrj.br, (gabriel, marcellogt)@dcc.ufrj.br

Abstract

In bioinformatics, DNA sequence assembly refers to the reconstruction of an original DNA sequence by the alignment and merging of fragments that can be obtained from several sequencing methods. The main sequencing methods process thousands or even millions of these fragments, which can be short (hundreds of base pairs) or long (thousands of base pairs) read sequences. This is a highly computational task, which usually requires the use of parallel programs and algorithms, so that it can be performed with desirable accuracy and within suitable time limits. In this paper, we evaluate the performance of DALIGNER long read sequences aligner in a system using the Intel Xeon Phi 7210 processor. We are looking for scalable architectures that could provide a higher throughput that can be applied to future sequencing technologies.

1 Introduction

In the very first years of DNA sequencing, mostly due to the technologies then available, the amount of data acquired in the sequencing processes had a very small volume and slow growth. Recently, however, new technologies for DNA sequencing with long reads have emerged, which are very promising in terms of assembly quality obtained, with much better results than short reads sequences methods.

This technology, on the other hand, must process files that contain reads with a high percentage of errors (up to 15 %) [8][13], requiring a large amount of repetitions of those reads and, consequently, a proportional increase in data volume and processing time.

One of the major challenges associated with DNA sequence assembly is the amount of time and computing resources required for this processing. Recent advances in computing systems resulted in more processing power, increased memory and data storage capacity, that assembly programs need to use in a more efficient way.

Parallel systems with high computational power, together with sequence assemblers using parallel techniques, are more and more used to process large amounts of data produced by DNA sequencers [1][4].

Intel Xeon Phi is a very cost-effective option to be used as a parallel system to perform DNA sequence assembly that has been made available in the recent years. This processor delivers

massive parallelism and vectorization with focus on high performance computing (HPC), which uses parallel processing for large data demands in a variety of areas, such as Computational Physics, Chemistry, Biology, and Finance [7].

The integrated and power-efficient architecture delivers significantly more compute per unit of energy consumed versus similar platforms, supplying an improved total cost of ownership.

2 Related Work

In recent years, very large genomes' assembly and alignment have improved with the development of new programs and sequencing technologies [6]. These programs are able to perform genome alignment and assembly with higher performance but using fewer computational resources.

BLASR (Basic Local Alignment with Successive Refinement) is an example of program used to aligning single Molecule Sequencing (SMS), using long read sequences [2].

BLASR combines the data structures used in short reads mapping with alignment methods used for whole genomes. The strategy used for mapping SMS reads is to locate a relatively small number of ranges where reads can be mapped and then detailed alignments are used to determine which one is the best range.

Another program is MECAT [9]. It uses an alignment method based on a different global alignment score. For large human SMS data, this method is 7 times more faster than MHAP [15] for paired alignment and 15 times more faster than BLASR using reference mapping. It is able to assembly large genome from single molecular sequencing (SMS) with high quality and low computing cost, using a smaller amount of memory and processor usage.

DALIGNER aligner for long read sequences finds local overlays and alignments in the datasets sequenced quickly and efficiently [6]. DALIGNER implementation process is divided into two steps. During the first step it looks for common k-mers present in the reference and target sequence. In the second step the alignment process is improved between the reference and the destination, to do this, DALIGNER uses an algorithm that is O(nd). In the next section more details about this will be presented.

These specific tools used in bioinformatics differ one from another not only in their algorithmic designs and methodology, but also in their performance robustness across a variety of datasets, processing time, memory usage and scalability [10].

As seen above, there are several studies published about DNA sequence assemblers evaluation, some focusing on quality and others addressing performance aspects.

The study presented here focus on evaluating DALIGNER performance using several execution parameters and compiler options, as well changing some internal configurations of the Intel Xeon Phi 7210 processor.

3 DALIGNER Aligner

DALIGNER can also be used as a general reading mapper, and a comparison tool between sequences, once "reads" can now be a DNA sequence. To find overlays, DALIGNER aligner uses a new adaptive computation method, based on an O(nd) algorithm described in [6], so that in practice an alignment is detected in linear time with the number of existing columns in the alignment.

DALIGNER is based on the same filtering concept adopted in BLASR, but using an optimized version of the radix sort algorithm [3]. The radix sort is an algorithm that uses counting

sort as a subroutine to sort. It sorts data with integer keys by grouping keys by the individual digits which share the same significant position and value. The second step improves the alignment between the reference and the destination.

It is necessary to split the dataset into smaller blocks to efficiently use DALIGNER on larger datasets. The comparisons required to perform all overlaps is quadratic in time relative to the number of blocks. DALIGNER optionally outputs full overlaps, but will first output local alignment trace points to aid in computing a full alignment in later steps, producing large auxiliary files [14].

4 Intel Xeon Phi

Knights Landing (KNL) is Intel's code name for its second-generation many-integrated-core (MIC) Xeon Phi processor, 7200 family. The change in hardware represents a significant improvement over the first-generation Knights Corner (KNC), giving the KNL the potential to be even more effective for memory-bound problems. The Table 1 shows some details and technical specifications of the Xeon Phi 7210 processor.

Table 1: Intel Xeon Phi System Technical Specifications

Intel Xeon Phi 7210	
Clock	1.30 GHz
CPU cores	64
Threads	256
Level 1 Cache	64 x 32 KB 8-way instruction 64 x 32 KB 8-way data
Level 2 Cache	32 x 1 MB 16-way shared
HBM memory	16 GB (Level 3 Cache)
Extensions	AVX-512 Vector Extensions
Physical memory	up to 384 GB
Performance	2.66 TFlops
TDP	215W

4.1 Memory Architecture

The Intel Xeon Phi KNL memory subsystem is composed of 16 GB accessed by 8 memory controllers, as well as up to 384 GB of DDR4 accessed by 2 x 3-channel memory controllers. An important aspect of the Knights Landing design, the achievable memory bandwidth is perhaps of the same or even greater importance. It is expected that the KNL chip can get more than 400 GB/sec of bandwidth out of the 16 GB of MCDRAM and more than 90 GB/sec out of the regular DRAM.

- **Flat Mode**: In this mode software modifications are required in order to use both the DDR and the MCDRAM in the same application. The MCDRAM memory is mapped into the same address space as the DDR memory and acts the same in terms of reading and writing. The advantage is that the 16GB of MCDRAM are seen as addressable, hence increasing the total addressable memory in a KNL system.

- **Cache Mode**: Using this mode does not require any software change and works well for many applications. In this mode cache is managed by the hardware and legacy applications will work just fine and can benefit from the high bandwidth memory (HBM).
- **Hybrid Mode**: This mode will use some of the MCDRAM as a cache, and some of it as flat memory. This is great for applications which can benefit from increased caching as well as take advantage of the higher bandwidth memory.

In Figure 1 one find the memory modes of the Intel XEON PHI. These modes are configured through the BIOS at boot time.

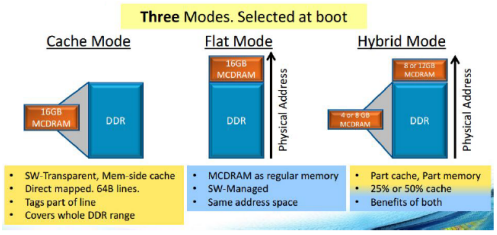

Figure 1: Memory Modes [11]

4.2 Clustering Modes

The KNL memory architecture has two types of memory: the high-bandwidth memory (HBM) integrated on package, MCDRAM, with capacity up to 16 GB and peak bandwidth over 450 GB/s and external DDR with capacity up to 384 GB (64 GB per channel) and peak bandwidth around 90 GB/s. The Knights Landing interconnecting mesh operates in one of three clustering modes: all-to-all, quadrant and sub-NUMA.

- **All-to-all**: Memory addresses are uniformly distributed across all tag directories in the chip. This is the most general mode with the easiest programming model.
- **Quadrant**: In the quadrant clustering mode, which divides the cores into into two (hemispheres) or four parts called quadrants and attempts to decrease intra-process communication time by keeping all threads of a single process close together.
- **Sub-NUMA**: Attempts to increase memory performance by keeping shared memory accesses to MCDRAM closer to the quadrant where the request originated. This mode provides the lowest latency, provided that applications are NUMA-aware.

5 Experimental Setup

Long read sequences obtained from Escherichia coli sequencing are used to perform the tests. It is also known as E. coli, and is a Gram-negative bacillary bacteria normally found in the intestines of humans and animals. Most E. coli strains are harmless; however, some are pathogenic, which means they can cause diseases that can be transmitted through contaminated food or water, or through contact with animals or people.

The dataset used to perform the tests was downloaded from the National Center for Biotechnology Information (NCBI) database at https://www.ncbi.nlm.nih.gov/nuccore/U00096.3. We

chose these data file because it is simple, very used in other works and our computing resources do not support a bigger dataset [14].

The sequence files are composed of fastq files captured from a PacBio RS II System and P6-C4 chemistry, the very first long read DNA sequencer [5]. A summary can be seen in Table 2.

Table 2: The size of data files

E. Coli str. K-12 substr. MG1655
700 GB

Since DALIGNER aligner does not use files in the fastq format, the seqtk [12] program was used to convert from the fastq to the fasta format.

It is possible to specify the k-mer size (k) to be used by DALIGNER. Three values for k-mer were used: the default value of k=14 and other two values, k=13 and k=15. These k-mer values were chosen because they presented the best hit count values. Some other k-mer values were also used, but presented very low hit count values, which indicates a poor alignment quality.

Two other parameters were set too: -M that specifies the maximum amount of memory to be used. As the system has 112 GB, this was fixed in 96 GB, so the program can be executed without any swap operation. Some other parameter defined was -T, specifying the amount of threads to be used. In the experiments performed this parameter was set between 1 and 60.

There is also a -t parameter which suppresses the use of any k-mer that has more than $t \times t$ occurrences in either the subject or target block. If the number set for -t is high, DALIGNER will need bigger memory to store the indexes of k-mer, if the number set for -t is low, DALIGNER will use less memory and storage but it could miss some alignments.

The best value for -t parameter depends on the size and repetition in the genome. If the -M parameter is used, the program automatically selects an effective value of -t that meets the memory limit specified [16]. We noticed that when k-mer values greater than 14 are used, the effective value of -t is set to its default value (100). But, when smaller k-mer values are chosen (e.g 13 or 12), the effective value of -t is reduced (e.g. 21 or 30) to fit into the memory limits specified.

The Intel VTune Amplifier XE 2017 tool was used to identify routines that consume more computational resources. It was observed that the Local_Alignment function consumes on average 81.5% of total execution time. The lex_thread, that performs k-mer sorting, is highly optimized and has a low impact (14.5%) in overall execution time.

Table 3: CPU usage by DALIGNER functions

Function	CPU Used (%)	Module
merge_thread	4%	daligner
lex_thread	14.5%	daligner
Local_Alignment	81.5%	daligner

The Local_Alignment module finds local alignments given a seed position, representing such local alignment with its interval and a set of pass-thru points, so that a detailed alignment can be efficiently computed on demand [6]. The Local_Alignment finds the longest significant local alignment between the sequences. A local alignment is specified by the point at which its path in the underlying edit graph starts, and the point at which it ends.

The Local_Alignment function has two routines: waves of f.r. (forward and reverse). The search takes place both in the forward direction and the reverse direction from a seed. The larger the size of the dataset, the greater will be the time spent in the function Local_Alignment over the other functions.

The majority of the tests were performed on a server with an Intel Xeon Phi 7210 accelerator and 112 GB (96 GB + 16 GB) of memory, and is called XEON PHI System for short. The files used in the experiment were stored on a local high-speed SSD (Solid-State Drive) disk. The operating system used was the version 7.2 of the 64-bit Centos Linux distribution and to compile DALIGNER it was used the Intel icc compiler version 17.0.4.

We also used a conventional multicore server, referenced as XEON system, whose characteristics are presented in the Table 4.

Table 4: XEON System Technical Specifications

Specifications	
Processor	Intel Xeon E5-2680 v2
Frequency	2.80 GHz
Cores	2 x 10
Cache	25 MB SmartCache
Memory	128 GB
Storage (SSD)	480 GB

For each result presented in this study, three series of tests were executed, the average time calculated and after that, the speedup and efficiency estimated.

6 Results

6.1 Execution Time Analysis

Initially DALIGNER was compiled and executed with the default parameters and environment variables found in the original makefile. On this first test, the XEON PHI system was configured with all-to-all cluster mode, flat memory mode. The results were named as DEFAULT configuration. Execution time results can be see in Figure 2.

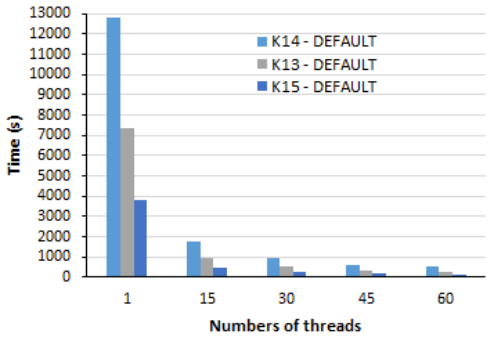

Figure 2: DEFAULT elapsed times

For comparison, DALIGNER was also compiled and executed on an conventional Intel Xeon server, using the default configurations and k=14.

Table 5 shows the execution time and speedup results using 1, 20 and 60 threads.

Table 5: Xeon versus Xeon Phi

Architecture	Elapsed Time			Speedup	
	1 Thread	20 Threads	60 Threads	20 Threads	60 Threads
Xeon	5833s	480s	677s	12.15	8.62
Xeon Phi	12840s	1350	540s	9.50	23.80

Although XEON PHI system had shown worse execution times initially, it outperforms the Xeon server when running 60 threads. This performance gain is mainly due to the higher number of cores (64) available in the Xeon Phi 7210 processor, and shows Xeon Phi architecture potential.

After running the DEFAULT tests, a new configuration called CONFIG1 was created, where the Makefile was modified, and the flag -xMIC-AVX512 and -O3 were added to the compiler options, and the environment variables KMP_AFFINITY and KMP_PLACE_THREADS were set to new values.

KMP_PLACE_THREADS controls allocation of hardware resources. For example, "64c,4t" specifies four threads per core on 64 cores, and "34c,2t" specifies two threads per core on 34 cores. The following value was attributed to variable KMP_PLACE_THREADS=64c,1t, that specifies one thread per core and using all cores available at the processor.

KMP_AFFINITY controls how threads are bound to resources. Common choices are COMPACT, SCATTER, and BALANCED. The granularity can be set to CORE or THREAD. Using the environment variable KMP_AFFINITY=scatter, it makes a round robin distribution of threads among the cores, so they are spread along the maximum number of cores.

To run this new configuration, the XEON PHI system was configured with all-to-all cluster mode and flat memory mode, i.e., the same values for the DEFAULT configuration.

With these changes, the execution times were 14% faster on average when compared to the DEFAULT configuration. In Figure 3 the results are shown.

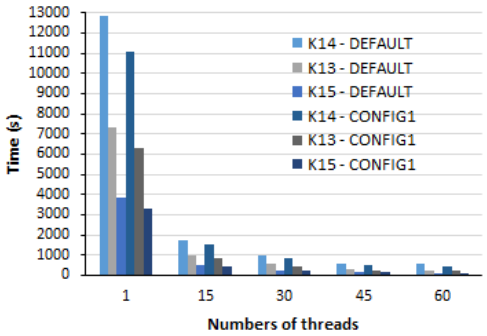

Figure 3: DEFAULT x CONFIG1 elapsed times

An additional configuration named CONFIG2 was made, with the same flags, optimization options and memory mode used with CONFIG1, but with the cluster mode changed to quad-

rant. In this case, the execution times were only 11% better than the results observed using the DEFAULT configuration, as shown in Figure 4.

Although it was expected a better performance using quadrant mode in comparison with all-to-all mode, this improvement did not occur. It was verified a performance loss within Local Alignment function. In this mode, this function consumes more computer resources than with all-to-all mode.

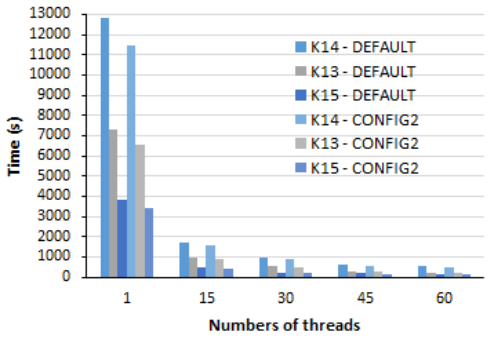

Figure 4: DEFAULT x CONFIG2 elapsed times

Two other additional tests were performed, changing memory mode to Hybrid and also to Cache. Using either these two settings, DALIGNER did not finish the alignment. Only the first step was completed, and when the second step was executed, the program crashes. We can not devise the reason for that behaviour, probably due to some flaw in the cache coherence protocol or race condition within the code.

6.2 Speedup and Efficiency Analysis

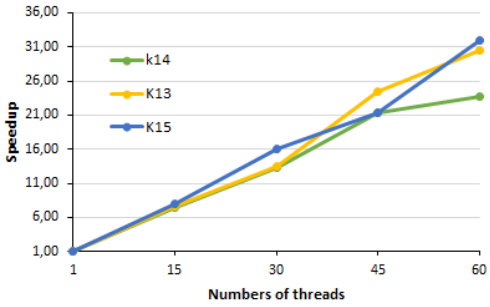

Figure 5: XEON PHI System speedup versus number of threads

The XEON PHI system presented a very consistent speedup, with an efficiency of 0.54 using 45 threads and k-mer value k=13. Figure 5 shows XEON PHI performance with three different k-mers values. As one can notice, there are not many differences between the speedups using k=13 and k=15. Table 6, summarize these results for values between 1 and 60 threads, using the DEFAULT configuration. Speedup and efficiency are roughly the same when using CONFIG1 and CONFIG2.

Table 6: Time, Speedup and Efficiency

K-mers	Threads	Time	Speedup	Efficiency
k=14	1	12840	1.00	1.00
	15	1740s	7.38	0.49
	30	960s	13.38	0.45
	45	600s	21.40	0.48
	60	540s	23.78	0.40
k=13	1	7320s	1.00	1.00
	15	960s	7.63	0.51
	30	540s	13.56	0.45
	45	300s	24.40	0.54
	60	240s	30.50	0.51
k=15	1	3840s	1.00	1.00
	15	480s	8.00	0.53
	30	240s	16.60	0.53
	45	180s	21.33	0.47
	60	120s	32.00	0.53

6.3 Memory Analysis

To assess the amount of memory consumed during the alignment process, the Linux *smem* command was used. This command shows how much physical memory is allocated for each running process. In Figure 6 the results of memory usage by the aligner on Intel XEON PHI system can be observed. The "-M 96" parameter was passed to the aligner, which limits the maximum of memory allocated to 96 GB.

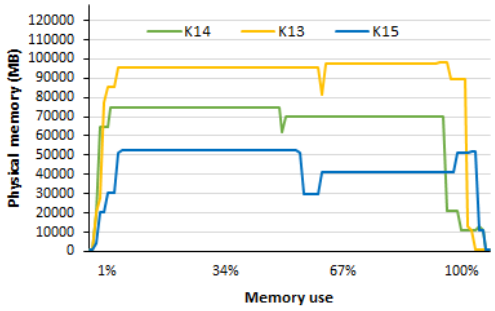

Figure 6: Memory footprint for DALIGNER. The x-axis shows the amount of memory used to run DALIGNER

As can be observed in Figure 6, as the k-mer size is increased, the memory footprint is smaller. This is highly expected, because large k-mers sizes reduce the total number of distinct k-mers and the amount of memory required to process them [16].

6.4 Alignment Quality

As much important as speedup and time measurements is the alignment quality. If the aligner has good speedup and time results, but a poor alignment quality, this is worthless. DALIGNER aligner output quality was verified through hit counting using k-mer sizes: k=14, k=13 and k=15 as seen in Table 7.

Table 7: Aligner Hits

k-mers	hit count	seed hits	confirmed hits
13-mers	2,767,529,738	6,529,247	8,499,596
14-mers	2,730,992,310	10,860,915	9,732,526
15-mers	1,439,546,906	9,283,118	9,589,822

As can be seen from Table 7, the best results for hit count, seed hits and confirmed hits were obtained with k-mer k=14. Using k=15 provides the lowest execution times and also better quality results than using k=13.

7 Conclusion and Future Work

In this study we presented some figures for DALIGNER performance on a system with Intel Xeon Phi 7210 processor. We made experiments with several configuration options, both in hardware and software, to evaluate the potential of this novel architecture executing a parallel genome assembler.

One of the contributions of this work is to quantify the differences in execution time, speedup and efficiency of an important bioinformatics application running on a system based on the Intel Xeon Phi 7210 processor. Although DALIGNER has not been designed to this type of architecture, it behaves pretty well according to our findings.

DALIGNER had a better performance executing on Xeon Phi, when compared to a conventional multicore processor. Although it has a lower clock frequency, this performance was achieved due to the higher number of cores (64 vs 20) available in Xeon Phi and the good scalibility attained.

Also, for several k-mer values tested, we concluded that k=14 remains a compromise between performance and assembly quality. Aligner quality is a very important information, since obtaining good speedup and time results but with poor alignment quality results is useless. As observed, when using the default k-mer value k=14, the best hit count results were achieved.

We also noticed that larger k-mers values use much less memory than smaller ones. As results, since we set a limit to the maximum amount of memory to be used, the quality of the assembly drops when using small k-mer values. The program automatically reduces the effective value of the -t parameter, The program automatically reduces the effective value of the -t parameter, resulting in some alignments loss.

Additionally, when tuning the architecture parameters for the Intel Xeon Phi, an improvement of 14% over the initial results was observed. Later, the configuration of the interconnecting mesh within the chip was modified, from All-to-All to Quadrant mode, resulting in a small performance loss, with execution times only 11% better than the initial results.

When using a specific feature of the Intel Xeon Phi 7210 as Memory Architecture, Clustering Modes and the environment variables KMP_AFFINITY and KMP_PLACE_THREADS,

DALIGNER posed a great potential for running on this architecture as seen in the results presented. With this we intend to help researchers to use computational resources based on Intel Xeon Phi architeture more efficiently. Although we use the 7210 model, the parameters used on the tests can be extended to others Intel Xeon Phi family processors.

As future work we intend to expand this work to other aligners such as BLASR, Minimap, Graphmap and MHAP, looking for a better tuning that includes hardware and software options, that can reduce the alignment execution time. Also we intend to investigate some others hardware and software options available to further improvements on DALIGNER performance.

Acknowledgment

The authors thank to Computer Science Department of Federal University of Rio de Janeiro for providing the computing resources used to conduct the experiments presented in this paper.

References

[1] Cantacessi, C., Campbell, B. E., Jex, A. R., Young, N. D., Hall, R. S., Ranganathan, S., and Gasser, R. B. Bioinformatics meets parasitology. Parasite Immunology, 34(5):265-275. 2012.

[2] Chaisson, M. J. and Tesler, G. Mapping single molecule sequencing reads using basic local alignment with successive refinement (blasr): application and theory. BMC Bioinformatics, 13(1):238. 2012.

[3] Cormen, T. H. Introduction to algorithms. MIT press. 2009.

[4] Costa, E. B., Silva, G. P., and Teixeira, M. G. Performance evaluation of parallel genome assemblers. In Saeed, F. and Haspel, N., editors, Proc. of the 7th Int. Conf. on Bioinformatics and Computational Biology (BICOB 2015), volume 1, pages 31-38. 2015.

[5] Eid, J., Fehr, A., Gray, J., Luong, K., Lyle, J., Otto, G., Peluso, P., Rank, D., Baybayan, P., Bettman, B., Bibillo, et al. Science, 323(5910):133-138. 2009.

[6] Myers, G. Efficient Local Alignment Discovery amongst Noisy Long Reads, pages 52–67. Springer Berlin Heidelberg, Berlin, Heidelberg. 2014.

[7] Rahman, R. Intel Xeon Phi Coprocessor Architecture and Tools: The Guide for Application Developers. Apress, Berkely, CA, USA, 1st edition. 2013.

[8] Sovic, I., Krizanovic, K., Skala, K., and Sikic, M. Evaluation of hybrid and non-hybrid methods for de novo assembly of nanopore reads. bioRxiv. 2015.

[9] Xiao, C.-L., Chen, Y., Xie, S.-Q., Chen, K.-N., Wang, Y., Luo, F., and Xie, Z. Mecat: an ultra-fast mapping, error correction and de novo assembly tool for singlemolecule sequencing reads. bioRxiv. 2016.

[10] Costa, Evaldo B.; Silvana C. Paulan . Processo de Sequenciamento e Montagem de Genomas. In: ERI-MT 2016 - VII Escola Regional de Informática de Mato Grosso, 2016, Rondonopolis - MT. ERI-MT 2016 - VII Escola Regional de Informatica de Mato Grosso, v. 1. p. 1. 2016.

[11] Codreanu, V., Rodríguez, S.J. Best Practice Guide - Knights Landing, Ole Widar Saastad (Editor), University of Oslo. 2017.

[12] Shen W, Le S, Li Y, Hu F. SeqKit: A Cross-Platform and Ultrafast Toolkit for FASTA/Q File Manipulation. PLoS ONE 11(10). 2016.

[13] Costa, Evaldo Bezerra. MELC Genomics: A Framework for De Novo Genome Assembly. Journal of Computational Biology, v. 1, p. cmb.2017.0102. 2017.

[14] Justin Chu,1,2 Hamid Mohamadi,1,2 René L Warren,2 Chen Yang,1,2 and Inanç Birol. Innovations and challenges in detecting long read overlaps: an evaluation of the state-of-the-art. Bioinformatics. Apr 15; 33(8): 1261–1270. 2017.

[15] Berlin K, Koren S, Chin CS, Drake PJ, Landolin JM, Phillippy AM Assembling Large Genomes with Single-Molecule Sequencing and Locality Sensitive Hashing. Nature Biotechnology. 2015.

[16] Chin C-S, Peluso P, Sedlazeck FJ, Nattestad M, Concepcion GT, Clum A, Dunn C, O'Malley R, Figueroa-Balderas R, Morales-Cruz A, et al. Phased diploid genome assembly with single molecule real-time sequencing. Nat Methods 13: 1050–1054. 2016.