Disk Partition Techniques Assesment and Analysis Applied to Genomic Assemblers Based on Bruijn Graphs

Nelson Enrique Vera-Parra[1], Ruben Javier Medina-Daza[2], Cristian Alejandro Rojas-Quintero[1]
[1] GICOGE Research Group, Distrital University Francisco José de Caldas, Carrera 7 No. 40B – 53, Bogotá D.C., Colombia.
[2] NIDE-GEFEM Research Group, Distrital University Francisco José de Caldas, Carrera 7 No. 40B – 53, Bogotá D.C., Colombia.

* Corresponding author. Tel.: +5713239300; email: nelsonenriquevera@gmail.com
Manuscript submitted December 1, 2016; accepted April 6, 2016.
doi: 10.17706/ijbbb.2016.6.2.59-67

Abstract: In this paper an assessment of several de-novo genomic assembler tools based on de Bruijn graph is made, with the purpose to measure the impact of the use of disk partitioning techniques regarding the computational requirements and generate a framework for bioinformatics researchers to let them identify advantages, disadvantages, bottlenecks and challenges of the assemblers using those techniques.

Assessed assemblers using disk partitioning techniques were: Minia and EPGA, the assessed assemblers that do not use disk partitioning were: ABySS and SOAPDenovo2. The parameters measured were the following: occupied space in RAM, processing time, parallelization and disk read and write access. A dataset was used with 36,504,800 short reads corresponding to 14th human chromosome. The assessment was made for two kmers size: 31 and 55. The results obtained were the following: The tools based on disk partitioning techniques showed the less RAM use. The tools with more I/O transfer intensity were the ones using disk partitioning techniques. The techniques that achieved more parallelization were the ones using disk partitioning.

Key words: Assemblers, assembly, bioinformatics, kmer count, minimizers.

1. Introduction

The rapid fall of sequencing costs since the arrival of the next generation sequencing (NGS) allowed the possibility to sequence complete genomes of different organisms with the aim to ease the study of their genes. One of the main problems and challenges is in the de-novo assembly stage [1], [2], in this stage short reads are used (between 100bp and 500bp) from machines with Illumina or Roche 454 technology and are assembled in contigs and metacontigs in such a way that as a result a complete assembled genome is obtained without need to count with a reference genome. For the assembly of small genomes, such as the bacterial genome assembly, these tools has good performance and most of the time require a little computation time. However as the genome size increases (eg the chromosome of a mammal) the computing requirements increases to perform this process [3].

Computational intensity required to perform an assembly without a reference genome (de-novo) has been addressed in the last decade using techniques based on De Brujin graphs [4]. Actually, the most used genomic assemblers are based on this approach, such as Velvet [5], ALLPATHS [6], ABySS [7], SOAPdenovo2 [8], MINIA [9].

Despite the remarkable results in the reduction of computing time for the assemblers based on De Bruijn, one of the aspects where high demand is still present is the memory use, mainly due the generation, storage and an kmer analysis and the graph representation. To mitigate these high memory requirement presented, in the last 5 years there have been assembler implementations that use disk partitioning techniques in some of its stages. Below is described the main techniques of disk partitioning used in the genomic assembly.

1.1. Kmers Distribution through a Hash Function

DSK [10] uses this method to make kmers partitioning in disk. To make the kmers disk partitioning is necessary to calculate the number of kmers, the number of iterations (*ni*) to cycle through the kmers and the number of partitions (*np*) to be created in disk. To make this calculation it must have input parameters such as the maximum memory use M (bits) and the kmer size k.

The hash function used in this algorithm is a function (*h*) that maps a kmer to a numeric value between [0 and H] where H is a big integer (2^64).

For each iteration, kmers (*m*) are taken one by one from the sequence. For each kmer the hash is calculated and the module operation is executed against the numer of iterations (*h(m) mod ni*), if the result of this operation is equal to the number of the actual iteration (*i*) the kmer is written to a file *Dj* where *j* is *h(m)/ni mod np*.

This method reduce significantly reduces the use of RAM in the kmers counting stage because it is not necessary to load all the kmers in memory. However, it may be the case that the kmers are not evenly distributed across all partitions.

1.2. Minimizers

The concept of minimizers was introduced for the first time in 2004 [11] with the purpose of facilitate the comparison of sequences because the method seed-and-extend is usually used for this task, this method requires a seed catalog that could take large amount of memory in the case of mammalian genomes or large proteins. To reduce the necessary space is required to save less kmers. The proposed method in [11] consist in choosing a representative kmer from a group of adyacent kmers such that different strings of text *Ti* and *Tj* have the same representative kmer if they share a large subsequence. A minimizer is a sub-chain (fixed size less than *k*) of a kmer with less lexicographical weight. However in a recent paper [12] the lexicographical weight is replaced by the frequence of the possible minimizers allowing to distribute more evenly the kmers on partitions.

In the case of assembly, inBCALM [12] the l-minimizer of a string u is the smallest l-mer that is a sub-string of u (assuming there is a total ordering of the strings, e.g. lexicographical). They define Lmin(u) (respectively, Rmin(u)) to be the l-minimizer of the (k − 1)-prefix (respectively suffix) of u. They refer to these as the left and right minimizers of u, respectively. They therefore perform in-memory l-mer counting to obtain a sorted frequency table of all l-mers. This step requires an array of 64|Σ| l bits to store the count of each l-mer in 64 bits, which is negligible memory overhead for small values of l (8 MB for l = 10). Each l-mer is then mapped to its rank in the frequency array, to create a total ordering of minimizers .

In previous works such as Assemblathon 1[13] Assemblathon 2 [14], and Genome Assembly Gold-Standard Evaluations (GAGE) [15] performance tests were conducted to the assemblers regarding to their results, that is, the number of generated contigs and their statistics. However in these works few mentions or none about the variables such as read/write operations, the RAM use and the parallelization of these assemblers. Moreover, some of the performance assessments available focus in the general performance analysis without focusing on each of its stages.

In this paper is proposed to assess several genomic assemblers based on De Bruijn graphs in order to

compare the computational demand of those that use disk partitioning techniques with respect to those that do not use them. For each assessed tool the data related to writing and reading, main memory use (RAM), paralellization (number and percentage of used processors) and processing time is measured with the purpose to verify which tools and which stages has major computational requirements.

2. Materials and Methods

2.1. Datasets

The dataset used to this assessment corresponds to 64587949 short reads (101bp), paired-end of the 14th Homo Sapiens chromosome.

2.2. Computational Equipment

The computer where this assessment was in place has the following described characteristics: Operating System: Debian Wheezy (AMD64). Processor: Intel(R) Xeon(R) cpu E7450 @ 2,40GHz, 24 cores. RAM: 64GB. Hard Disk: 160GB HDD.

2.3. Assessed Assemblers

The tools to evaluate are assemblers based on de bruijn graphs that use only short paired-end reads for contigs determination. Assemblers were chosen in such a way that there was diversity of kmercounting algorithms and the graph representation structures in memory, additionally as a selection criterion was taken the major use during the last decade and its lastest version was published in the year were thar assesment was performed (2015). The tools to evaluate were grouped according to the use or not of disk partitioning techniques. Note that assemblers such as ALL-PATHS-LG were not included in this assessment because it requires long fragments as additional to the short fragments to be executed.

2.3.1. Assessed assemblers that does not use disk partitioning techniques

ABySS: Is a de novo parallel assembler for paired sequences, specifically designed for short reads. Roughly the algorithm used by this tool has the following stages: First, without using the paired-end information, contigs are extended until either they cannot be unambiguously extended or come to a blunt end due to a lack of coverage. In the second step the paired-end information is used to resolve ambiguities and merge contigs [7]. ABySS is implemented in C++ and use the *Openmpi* library (in some of its stages) with the purpose to allow the comunication bewteen several computer nodes and ease its execution on a cluster. Additionally ABySS use the google-sparsehash library to decrease the RAM requirement in the kmers counting.

SOAPdenovo2: Is a novel short-read assembly method that can build a de novo draft assembly for the human-sized genomes. This algorithm has the following steps: De Bruijn Graph (DBG) construction, contig assembly, paired-end (PE) reads mapping, scaffold construction, and gap closure. SOAPdenovo2 is implemented in C++ and for some of its processing stages use several threads.

2.3.2. Assessed assemblers that use disk partitioning techniques

Minia: Is a short-read assembler based on a de Bruijn graph, capable of assembling a human genome on a desktop computer in a day. The output of Minia is a set of contigs. The main stages for the Minia algorithm are: kmer counting, graph construction and the cycle or simplification of it. For the kmer counting stage, Minia uses DSK that use the distribution on disk of kmers method through a hash function with the purpose to reduce the RAM memory use. Minia is implemented on C++. Note that as opposed to Abyss, Minia does not use the available paired-end information, this means that it does not create metacontigs.

EPGA2: Is an genomic assembler oriented to efficient memory use. To accomplish this purpose it uses BLESS [16] for error correction in the reads, the kmers distribution in disk method through the hash

function with the use of DSK [10] for the kmer counting stage and the minimizers method through BCALM [12] for node simplification in the De Bruijn graph. Once the nodes are simplified and contigs generated EPGA2 joins the contigs in parallel [17].

3. Results

The Fig. 1 and the Fig. 2 show the RAM use a peak for each assembler and its stages, for parameters *k*=31 and *k*=55 respectively.

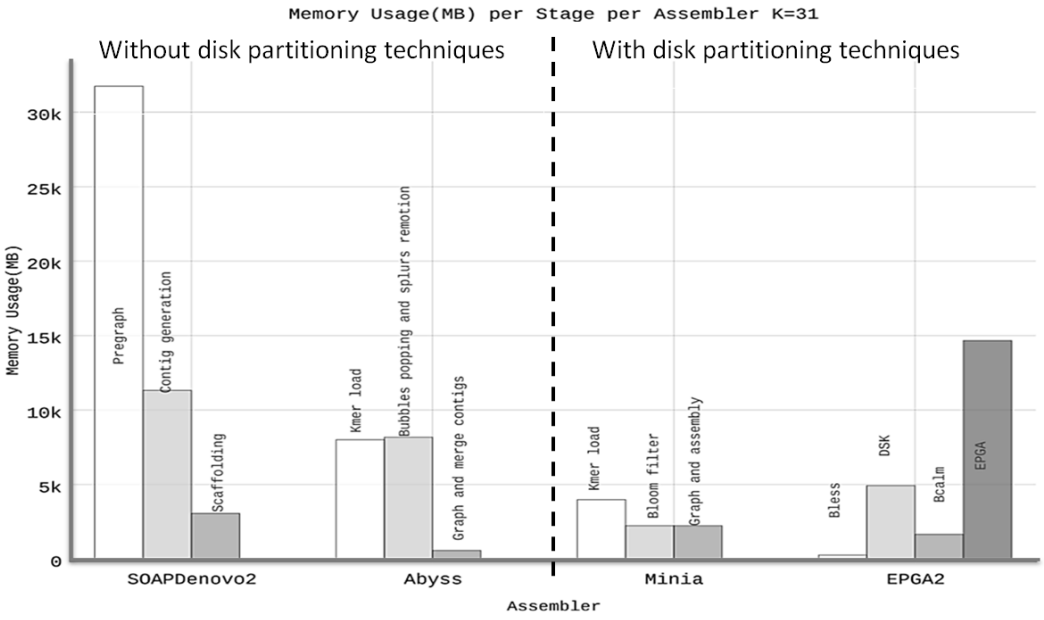


Fig. 1. RAM use peaks for each tool (*k*=31).

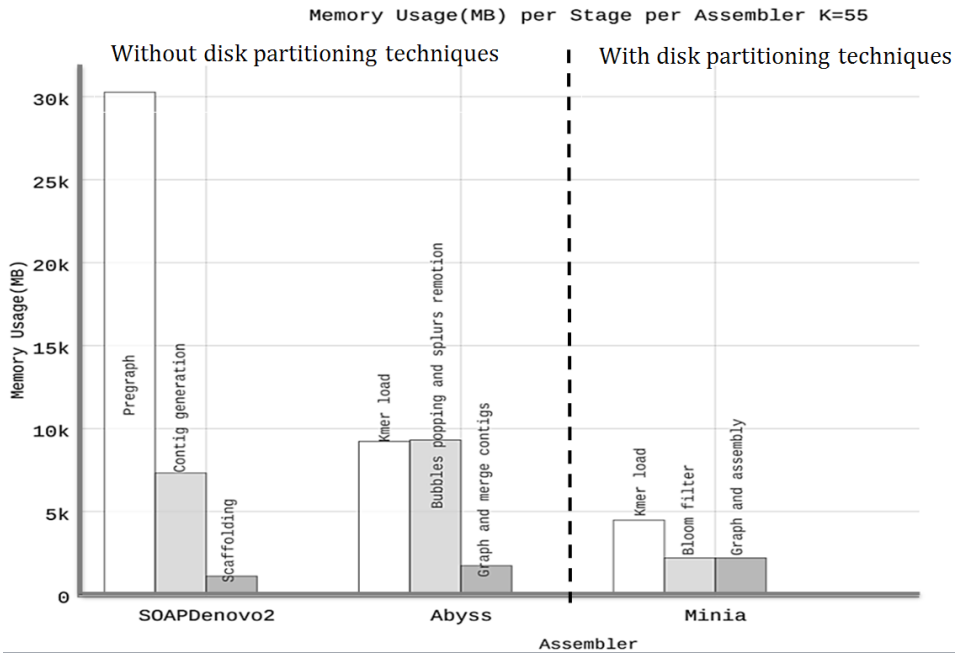


Fig. 2. RAM use peaks for each tool (*k*=55).

The Fig. 3 and Fig. 4 show the CPU usage number peak and average for each assembler and its stages for parameters *k*=31 and *k*=55 respectively.

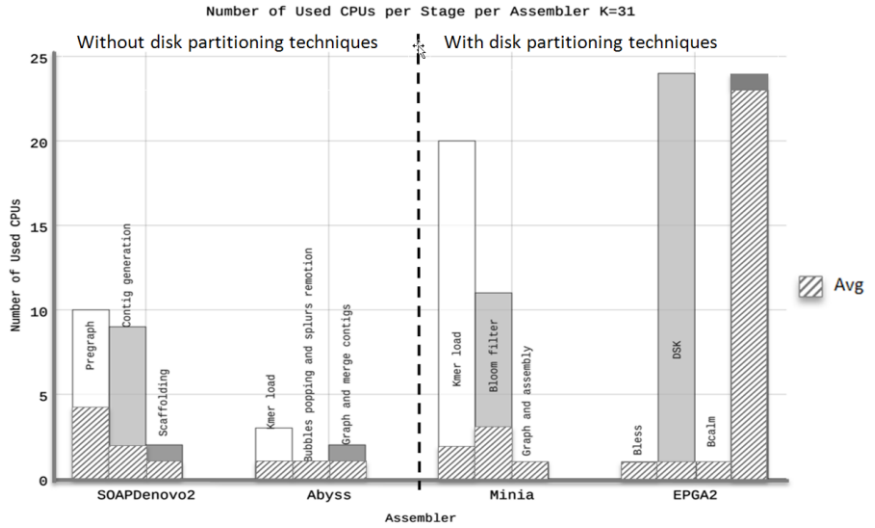


Fig. 3. CPUs use peaks for each tool (k=31).

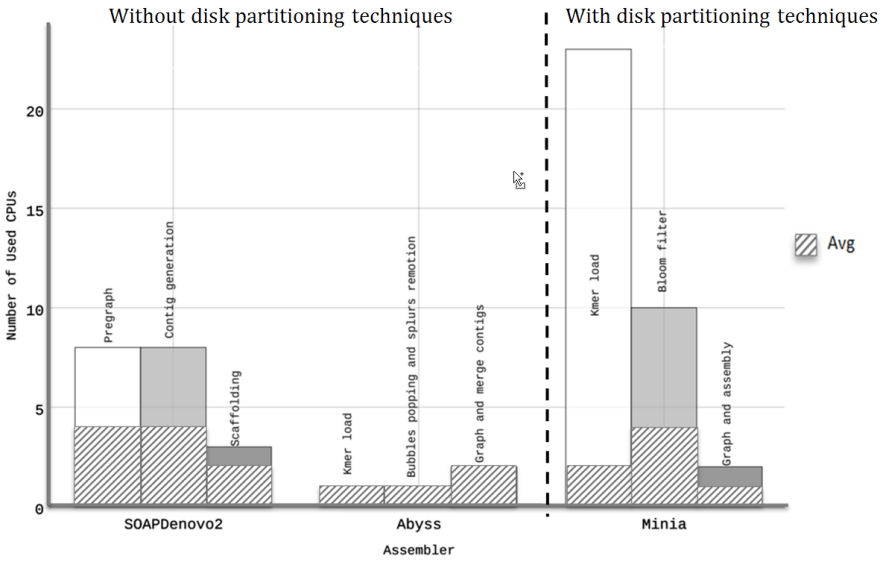


Fig. 4. CPUs use peaks and averages for each tool (k=55).

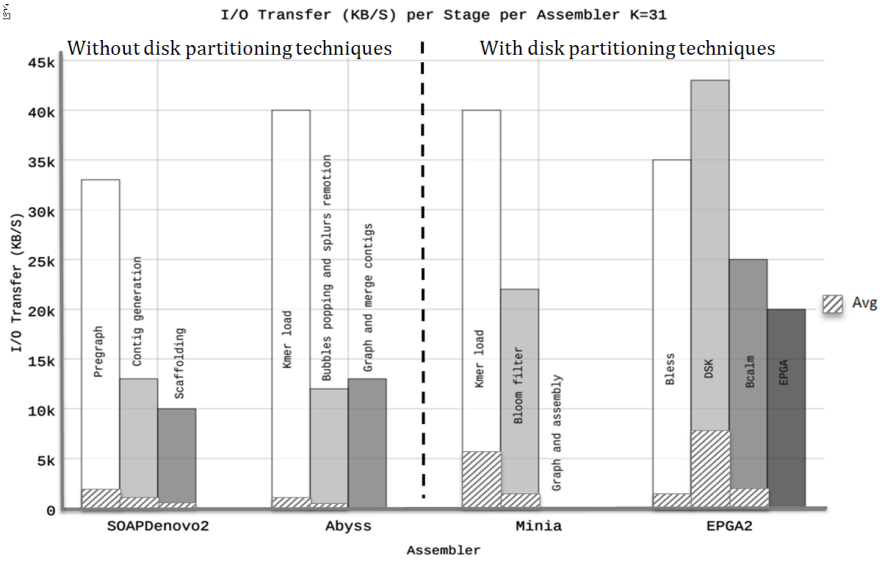


Fig. 5. Peaks and average I/O transfer for each tool (k=55).

The Fig. 5 and Fig. 6 show the peak and average I/O transfer for each assembler's stages and its stages for the parameter *k*=31 and *k*=55 respectively.

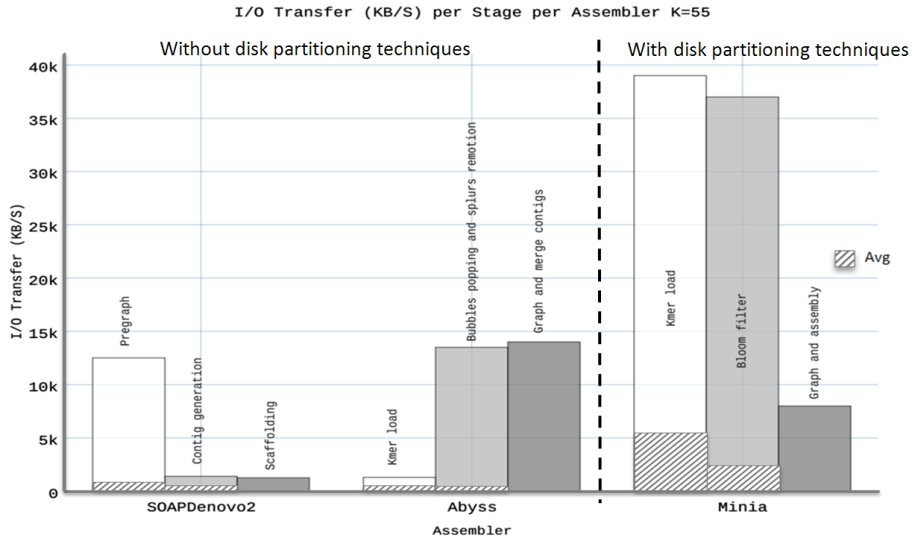


Fig. 6. Peaks and average I/O transfer for each tool (*k*=55).

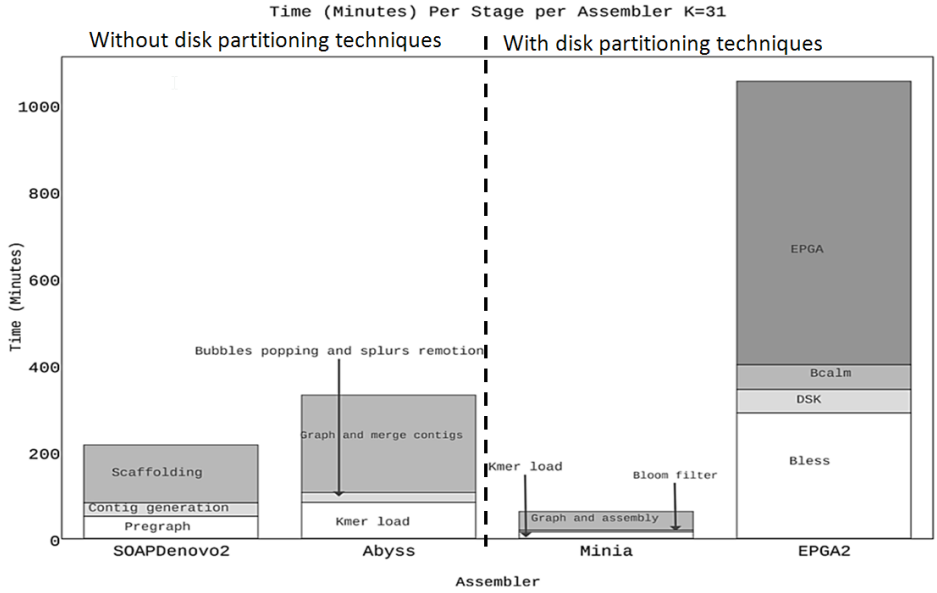


Fig. 7. Execution time of each tool (*k*=31).

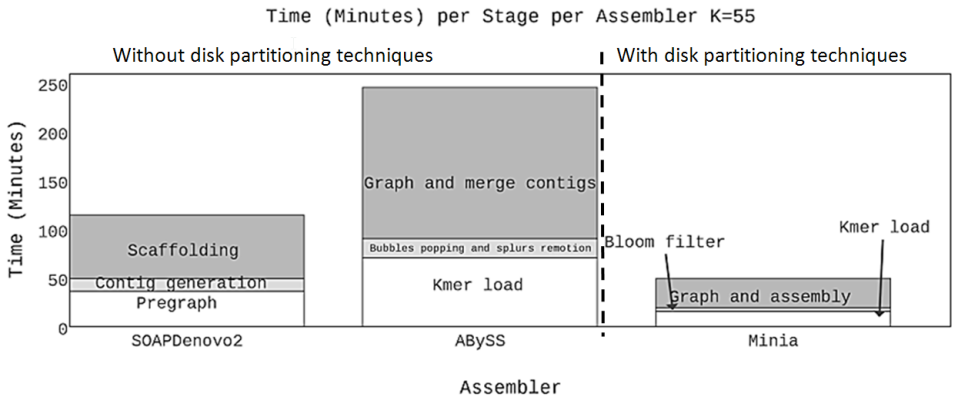


Fig. 8. Execution time of each tool (*k*=55).

Fig. 7 and Fig. 8 show the execution time used by each stage for each assembler for the parameters *k*=31 and *k*=55 respectively.

4. Conclusions.

Regarding the RAM use it can be concluded that for the counting kmers stage the tools with the less RAM use were the ones using disk partitioning techniques through a hash function, for example, Minia and EPGA2 using DSK. The tool with the less RAM use in the contigs generation stage was EPGA, since it makes use of disk partitioning by minimizers technique (selected by frequency) in the stage of contigs generation (BCALM).

Regarding the I/O transfer it was noted that the tools that use disk partitioning techniques had the highest average of transfer in the counting kmer stage. Additionaly for the contigs generation stage the tool with the highest average I/O transfer was EPGA, because of the use of BCALM to simply nodes in the graph.

As for paralellization it can be noted that the tools that use disk partitioning have greater paralellization in the counting kmers stage.

Regarding the execution time, the tools that use disk partitioning took the less execution time in the counting kmers stage, however in the metacontigs generation stage EPGA required a greater amount of time.

Acknowledgments

Work done in collaboration with High Performance Computational Center (CECAD) - Distrital University Francisco José de Caldas, Bogotá D.C., Colombia (http://cecad.udistrital.edu.co) and Genetics Institute - National University (IGUN), Colombia, (http://www.genetica.unal.edu.co).

References

[1] Miller, J. R., Koren, S., & Sutton, G. (2010). Assembly algorithms for next-generation sequencing data. *Genomics, 95(6)*, 315-327.

[2] Vera-Parra N., Perez-Castillo, J., & Rojas-Quintero, C. (2015). Performance assessment for main stages in genomic and transcriptomic data processing based upon reads from illumina sequencing technologies. *International Journal of Applied Engineering Research (IJAER)*, 34670-34674.

[3] Zhang, W., Chen, J., Yang, Y., Tang, Y., Shang, J., & Shen, B. (2011). A practical comparison of de novo genome assembly software tools for next-generation sequencing technologies. *PloS one, 6(3)*, e17915.

[4] Pevzner, P. A., Tang, H., & Waterman, M. S. (2001). An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences,* Vol. 98, No. 17, pp. 9748-9753.

[5] Zerbino, D. R., & Birney, E. (2008). Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. *Genome research, 18(5)*, 821-829.

[6] Butler, J., MacCallum, I., Kleber, M., Shlyakhter, I. A., Belmonte, M. K., Lander, E. S., & Jaffe, D. B. (2008). ALLPATHS: De nsdovo assembly of whole-genome shotgun microreads. *Genome research, 18(5)*, 810-820

[7] Simpson, J. T., Wong, K., Jackman, S. D., Schein, J. E., Jones, S. J., & Birol, I. (2009). ABySS: A parallel assembler for short read sequence data. *Genome Research, 19(6)*, 1117-1123.

[8] Luo, R., Liu, B., Xie, Y., Li, Z., Huang, W., Yuan, J., & Wang, J. (2012). SOAPdenovo2: An empirically improved memory-efficient short-read de novo assembler. *Gigascience, 1(1)*, 18.

[9] Chikhi, R., & Rizk, G. (2013). Space-efficient and exact de Bruijn graph representation based on a bloom filter. *Algorithms for Molecular Biology, 8(22)*, 1.

[10] Rizk, G., Lavenier, D., & Chikhi, R. (2013). DSK: k-mer counting with very low memory usage.

Bioinformatics, btt020.

[11] Roberts, M., Hayes, W., Hunt, B. R., Mount, S. M., & Yorke, J. A. (2004). Reducing storage requirements for biological sequence comparison. *Bioinformatics, 20(18)*, 3363-3369.

[12] Chikhi, R., Limasset, A., Jackman, S., Simpson, J. T., & Medvedev, P. (2014, January). On the representation of de Bruijn graphs. *Research in Computational Molecular Biology*, 35-55.

[13] Earl, D., Bradnam, K., John, J. S., Darling, A., Lin, D., Fass, J., & Xie, Y. (2011). Assemblathon 1: A competitive assessment of de novo short read assembly methods. *Genome Research, 21(12)*, 2224-2241.

[14] Bradnam, K. R., Fass, J. N., Alexandrov, A., Baranay, P., Bechner, M., Birol, I., & MacCallum, I. (2013). Assemblathon 2: Evaluating de novo methods of genome assembly in three vertebrate species. *GigaScience, 2(1)*, 1-31.

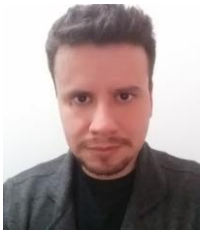
[15] Salzberg, S. L., Phillippy, A. M., Zimin, A., Puiu, D., Magoc, T., Koren, S., & Yorke, J. A. (2012). GAGE: A critical evaluation of genome assemblies and assembly algorithms. *Genome research, 22(3)*, 557-567.

[16] Heo, Y., Wu, X. L., Chen, D., Ma, J., & Hwu, W. M. (2014). BLESS: Bloom filter-based error correction solution for high-throughput sequencing reads. *Bioinformatics, 30(12),* 1354-1362.

[17] Luo, J., Wang, J., Li, W., Zhang, Z., Wu, F. X., Li, M., & Pan, Y. (2015). EPGA2: Memory-efficient de novo assembler. *Bioinformatics, 31(24)*, 3988-3990.

Cristian Alejandro Rojas Quintero was born in Bogota, Colombia in March 1992. He is systems engineer from District University Francisco José de Caldas (UDFJC) (2015). He is also a master student in information science and communications at UDFJC.

He is research assistant at UDJFC, working for research center and scientific development (CIDC). Some of his publications includes: Biopython Basic: Practice Manual (Bogotá, Colombia: District University Editorial, 2014), The immunotranscriptome of the Caribbean reef-building coral Pseudodiploria strigosa (Germany: Inmunogenetics, 2015), Rna-seq Ud: Una Plataforma Bioinformática Para Análisis Rna-seq (Braga, Portugal: Atas da 10º Conferência Ibérica Sistemas E Tecnologias De Informação). He is currently researching in genome assembly processes and heterogeneous computing.



Nelson Enrique Vera Parra was born in Ibagué, Colombia in December 1979. He received his master in information science and communications from District University Francisco José de Caldas (UDFJC), Bogotá, Colombia in 2008. He is electronic engineer from the South Colombian University, Neiva, Colombia from 2002. He is also PhD. student in UDFJC.

He is currently a titular professor at UDFJC, working for the Faculty of Engineering (Electronic Engineering). Some of his publication includes: Biopython Basic: Practice Manual (Bogotá, Colombia: District University Editorial, 2014) , Optimización del preprocesamiento de lecturas de secuenciación de nueva generación (Bogotá, Colombia: Redes De Ingeniería, 2014), The immunotranscriptome of the Caribbean reef-building coral Pseudodiploria strigosa (Germany, Inmunogenetics, 2015).He is currently researching in genome assembly processes and heterogeneous computing.

Rubén Medina Javier Daza was born in Bogota, Colombia in March 1979. He is a PhD in computer Science with emphasis in geographic information systems (GIS). He received his master's degree in GIS from UPSAM (Spain) in 2008 and also a master's in teleinformatics from District University Francisco José de Caldas (UDFJC), Bogotá, Colombia in 2002. Other degrees include specialist in software engineering (1997), specialist in GIS (2005) and mathematics from UDFJC (1995).

He is currently a titular professor at UDFJC, working for the Faculty of Engineering (Geodesy and Cadastral Engineering) and for the master of science in information and communications program. Some of his publications includes: Implementing Fast-Haar Wavelet transform on original Ikonos images to perform image fusion: qualitative assessment (Medellín, Colombia: Antioquia University Editorial, 2014), Aplicativo Web para la Fusión de Imágenes Satelitales (Oporto, Portugal: Risti - Revista Ibérica De Sistemas E Tecnologias De Informação, 2013), Matemáticas especiales. Una aplicación de la transformada de Fourier en el análisis de imágene (Bogotá, Colombia: District University Editorial, 2013). He is currently researching in mathematical computing.