computing@computingonline.net
www.computingonline.net

Print ISSN 1727-6209
On-line ISSN 2312-5381
International Journal of Computing

HETEROGENEOUS COMPUTING TO ACCELERATE THE SEARCH OF SUPER K-MERS BASED ON MINIMIZERS

Nelson Enrique Vera-Parra, Danilo Alfonso López-Sarmiento, Cristian Alejandro Rojas-Quintero

Francisco José de Caldas District University, Bogotá, Colombia
neverap@udistrital.edu.co, dalopezs@udistrital.edu.co, carojasq@correo.udistrital.edu.co,
http://gicoge.udistrital.edu.co/

Paper history:
Received 12 May 2020
Received in revised form 25 August 2020
Accepted 17 October 2020
Available online 30 December 2020

Keywords:
Heterogeneous computing;
K-mers processing;
OpenCL;
Parallel programming;
Search and distribution of super k-mers.

Abstract: The k-mers processing techniques based on partitioning of the data set on the disk using minimizer-type seeds have led to a significant reduction in memory requirements; however, it has added processes (search and distribution of super k-mers) that can be intensive given the large volume of data. This paper presents a massive parallel processing model in order to enable the efficient use of heterogeneous computation to accelerate the search of super k-mers based on seeds (minimizers or signatures). The model includes three main contributions: a new data structure called CISK for representing the super k-mers, their minimizers and two massive parallelization patterns in an indexed and compact way: one for obtaining the canonical m-mers of a set of reads and another for searching for super k-mers based on minimizers. The model was implemented through two OpenCL kernels. The evaluation of the kernels shows favorable results in terms of execution times and memory requirements to use the model for constructing heterogeneous solutions with simultaneous execution (workload distribution), which perform co-processing using the current search methods of super k -mers on the CPU and the methods presented herein on GPU. The model implementation code is available in the repository: https://github.com/BioinfUD/K-mersCL.

1. INTRODUCTION

The search of super k-mers of a genomic read is a task that demands finding the seed (canonical minimizer or signature) of each possible k-mer and compare them with each other in order to identify those contiguous k-mers that have the same minimizer [1]. This search becomes an intensive task when it must be performed for millions of reads due to the high number of processes and the large amount of both input and output data. Due to the independence of processes between reads, the search for super k-mers is a highly suitable task to be accelerated by simultaneous heterogeneous processing: the workload is partitioned to be processed simultaneously between the CPU and the GPU(s), either through a static, dynamic [2], or hybrid distribution [3]. However, for this type of processing to be carried out efficiently it is necessary to overcome the following challenge: the search for super k-mers is a process that has a very high and unpredictable memory requirement when it is massively parallelized because the space required depends on the data generated but not on the input data. The memory occupied by the super k-mers of a read depends on the sequence (it varies from read to read) and is much greater than the space occupied by the read due to redundancy. This prevents the search of super k-mers from running efficiently on many-core platforms such as GPUs where the memory is limited and the cost of data transfer between the levels of the memory hierarchy is very high.

This paper proposes a massive model of parallel processing for the search of super k-mers that allows the memory requirements and the execution times to be adequate to develop efficient heterogeneous solutions with simultaneous CPU-GPUs execution. The following section provides a background to the thematic framework of the problem, in section 3 the components that constitute the massive parallel processing model are presented: a heterogeneous processing model, an efficient data structure and 2 parallelization patterns; finally, the results are presented and analyzed, conclusions are drawn and future works are proposed.

2. BACKGROUND

2.1 WHAT ARE MINIMIZERS?

Minimizers were initially proposed as a technique to reduce storage requirements in biological sequence comparison processes [4, 5], through the strategy of reducing the redundancy presented by the “seed-and-extend” technique [6]. As of 2013, the minimizers ventured into k-mers treatment tasks in de-novo assembly processes: (a) k-mers counters [7] KMC2 and KMC3 [8] and MSPKmerCounter [9], (b) graph builder MSPGraphBuilder [10], (c) graphs compactors [11,12], and (d) assembler EPGA2 [13]. A minimizer of a k-mer is the sub-sequence of length m (m-mer, where m < k); when comparing all possible m-mers of that k-mer, this minimizer presents the lowest value according to a criterion of comparison, which is usually the lexicographic weight.

According to the previous definition, the minimizer of a k-mer is a unique sub-sequence, and those k-mers that are contiguous in a read are very likely to present the same minimizer. For this reason they can be considered “seeds” and used as a partitioning criterion and as a data structure (super k-mers: merging contiguous k-mers with the same minimizers).

2.2 WHAT DO THE MINIMIZERS CONTRIBUTE TO THE PROCESSING OF K-MERS?

Minimizer are used to face the two challenges of processing k-mers: the high volume of data due to redundancy and the impossibility or difficulty of partitioning treatment [7].

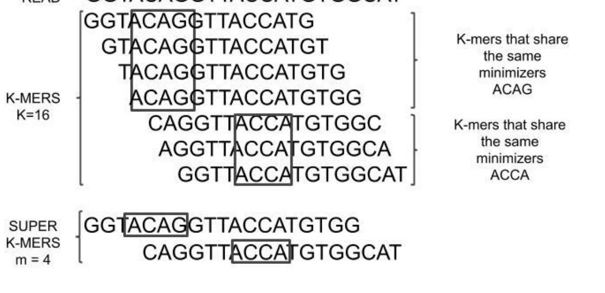

Figure 1 – Super k-mers. Data structure for reducing redundancy

Data structure to reduce redundancy: Minimizers are used to define data structures where not all the k-mers of a read are stored, but those that are contiguous and have the same minimizer are merged [14]. The product of this fusion is sub-sequences called super k-mers [8]. Fig. 1 shows how to go from 7 k-mers (112 bases) to 2 super k-mers (37 bases).

Criterion for partition of the data set: Although the merger of k-mers in super k-mers reduces redundancy, when dealing with said super k-mers it is very likely that the process needs to obtain all its k-mers; this would demand the same memory as if all the k-mers of the readings of the data set had been obtained and stored initially. For this reason, minimizers are also used as a criterion to divide the data set by creating partitions (files on disk) that contain all the super k-mers that have the same minimizer, so that these files can be stored on memory and process separately by ensuring that the same k-mer will not exist in a different partition and that all contiguous k-mers that have the same minimizers will be in the same partition (see Fig. 2).

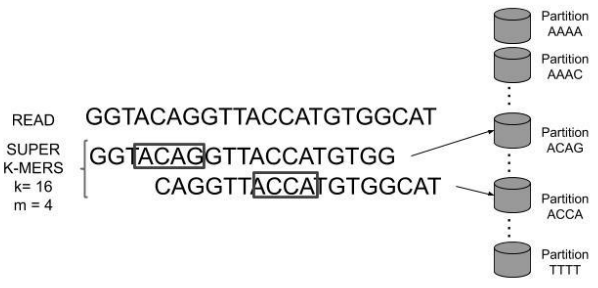

Figure 2 – Minimizer. Criterion for partition of the data set

2.3 WHAT ARE THE CANONICAL MINIMIZERS?

The partitioning technique is performed to execute a type of processing that takes into account the reverse complement of the sequences and sub-sequences [15, 16]. The minimizer criterion could not be applied as explained above, because a k-mer could remain in one partition and its reverse complement in another. To solve this, a small modification is made to the selection criterion of the minimizer in such a way that the one with the lowest lexicographic weight is chosen, but in this case not only the m-mers but also its reverse complements are included.

2.4 WHAT ARE THE SIGNATURES?

If the criterion for selecting the minimizers is exclusively the lexicographic weight of the m-mers, the sizes of each of the partitions will depend directly on the frequency of occurrence of the minimizers. This means that they are defined by the trends in the biological sequences. For example, minimizers with several As can be very common, while minimizers with several Ts can be very rare. This could lead to non-uniform distributions that generate large and small files that do not optimize the use of memory. In order to achieve more homogeneous distributions, the authors of the k-mers counting tools KMC 2 and KMC 3 proposed a canonical minimizer alternative where some of them are vetoed according to specific characteristics. Therefore, those allowed minimizers have not so different frequencies of occurrence. Allowed minimizers are those that meet the following characteristics: they do not start with AAA or ACA and do not contain AA anywhere (except at the start).

3. MATERIAL AND METHODS

3.1 HETEROGENEOUS PROCESSING MODEL

This model is supported in three pillars: (1) a data structure called CISK (Compact-Indexed Super k-mers representation) for representing the super k-mers and their seeds in a compact and indexed way, so the identification process of super k-mers can be carried out without performing the process of explicit extraction of the super k-mers, and (2,3) two massive parallelization algorithms for obtaining the canonical m-mers and identifying the super k-mers of a set of readings through the efficient use of many-core devices. The purpose of the proposed model and its three contributions is to facilitate the development of simultaneous heterogeneous solutions: current efficient search processes of super k-mers executed in parallel on CPU, accelerated by simultaneous co-processing on GPU(s) (see Fig. 3).

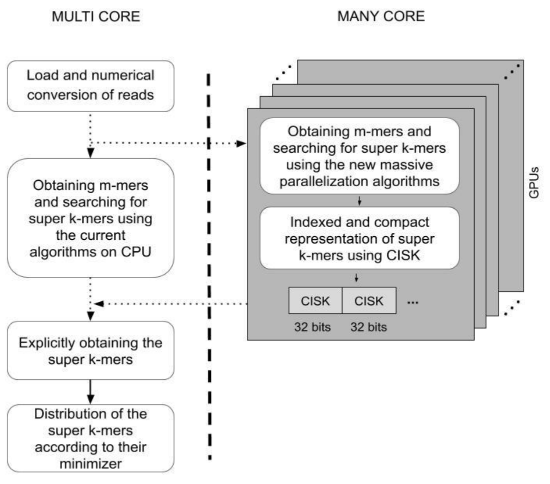
MULTI CORE
MANY CORE
Load and numerical conversion of reads
Obtaining m-mers and searching for super k-mers using the current algorithms on CPU
Explicitly obtaining the super k-mers
Distribution of the super k-mers according to their minimizer
Obtaining m-mers and searching for super k-mers using the new massive parallelization algorithms
Indexed and compact representation of super k-mers using CISK
CISK
CISK
...
32 bits
32 bits
GPUs

Figure 3 – Heterogeneous processing model with simultaneous CPU-GPUs execution

3.2 CISK (COMPACT-INDEXED SUPER K-MERS REPRESENTATION)

The model proposes a data structure for representing the super k-mers and their minimizers in an indexed and compact way, so they are detected and stored in the GPU without extracting them explicitly. Each super k-mer of a read is represented by three data (minimizer, position of the super k-mer, and length of the super k-mer minus k) that are compacted in only 32 bits (see Fig. 4). The 7 least significant bits are used to represent the number of k-mers that are added to the initial k-mer to build the super k-mer (the length of the super k-mer minus k), so the maximum length of super k-mer is k + 2 ^ (7) and the maximum k is 2 ^ 7 + m. The remaining 25 bits are shared to represent the seed (canonical minimizer or signature) and the initial position of the super k-mer within the read. The distribution of these 25 bits is performed dynamically as a function of m: the 2 * m most significant bits are reserved to represent the seed and the rest are assigned to represent the position of the super k-mer. In this way the maximum read length supported is 2 ^ (25 - 2 * m) + k - 1. For example, with m = 4 the length of the reads can be up to 131,072 + k - 1 (PacBio reads [17, 18]), and with m = 9 it can be up to 128 + k (typical short reads). Note: The use of small m generates few partitions that could cause an unbalanced distribution; in this case the use of signatures is recommended in order to reduce this problem.

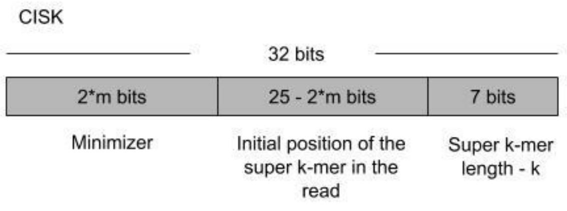

Figure 4 – CISK. Compact-Indexed Super k-mers representation

3.3 MASSIVE PARALLELIZATION ALGORITHM: OBTAINING CANONICAL M-MERS OF A SET OF READS

A canonical m-mer is the smallest between the m-mer and its reverse complement taking its lexicographic weight as a comparison value. The conventional way to find these weights is by means of its base-10 numerical representation. The decimal values could be calculated in a relatively simple way on an ideal many-core device (infinite physical threads, unlimited memory, and non-latency) by assigning a thread to compute each base. However, an implementation of this form on a real device would lead to an over-access of each of the bases of the read up to m-1 times, which would be catastrophic for the operational intensity. In addition, it would have overly fine granularity that would be inefficient considering the way the threads are executed. As a consequence, this model proposes a new parallelization algorithm for obtaining the canonical m-mers of a set of reads based on tiles with hybrid granularity, in such a way that a massive parallelism (m threads per tile) can be used to obtain the first m-mer of each tile (by means of conventional numerical conversion equations) and a moderate parallelism (1 thread per tile) to obtain the rest of m-mers by means of a roll strategy that allows the reuse of the results, thereby avoiding redundancy in the access to each element of the input vector (see "kernels algorithms" in the repository: https://github.com/BioinfUD/K-mersCL).

3.4 MASSIVE PARALLELIZATION ALGORITHM: SEARCH FOR SUPER K-MERS BASED ON MINIMIZERS

The search process of the super k-mers of a read demands the analysis of all its k-mers to identify those that are contiguous and have the same minimizer. This model proposes a sequential / parallel strategy that significantly reduces the number of k-mers to be analyzed and the number of threads to be used: through a sequential process (window by leaps) the zones that contain the boundaries between two super k-mers are identified, and through a parallel process these zones are analyzed to accurately detect the boundaries according to the minimizers (see Fig. 5).

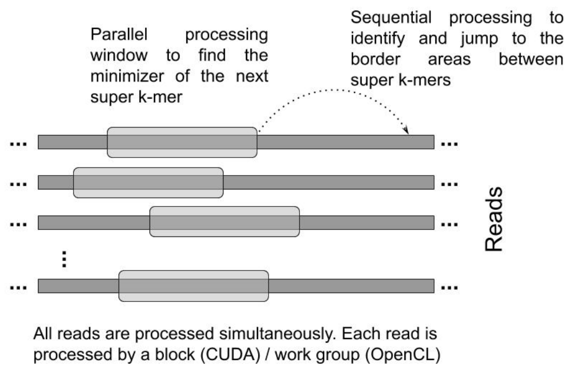
Parallel processing window to find the minimizer of the next super k-mer
Sequential processing to identify and jump to the border areas between super k-mers
Reads
All reads are processed simultaneously. Each read is processed by a block (CUDA) / work group (OpenCL)

Figure 5 – Sequential / parallel strategy. Search for super k-mers using GPUs

If the zone has a reference minimizer, a pattern that finds the closest lowest canonical m-mer using atomic operations will be used; this pattern will be the minimizer of the new super k-mer. If the zone does not have a reference minimizer, a mixed and adaptive reduction pattern will be used to find the new minimizer. This pattern is mixed because it uses a three-level tree structure, but its branches converge through atomic operations, and it is adaptive because the number of branches of the second level (to which the initial elements converge) is adapted according to the size of the zone. Then the number of elements per block to be reduced in each level slightly varies and has low levels (around 8). The atomic operations therefore have high and homogeneous performance in each of the tree levels [19].

It is possible that several elements of the zone have the same minimum value. In that case, the one with a greater position is selected in order to favor the construction of more extensive super k-mers (see "kernels algorithms" in the repository: https://github.com/BioinfUD/K-mersCL).

3.5 IMPLEMENTATION AND EVALUATION

The massive parallel processing model for the search of super k-mers was implemented through two OpenCL [20, 21] kernels, one for canonical minimizers and the other for signatures. In order to facilitate testing and evaluation of the kernels, a host code was implemented using PyOpenCL [22, 23]; this code performs the complementary tasks to search the super k-mers (load and numerical conversion of reads and explicit extraction of super k-mers). Both the kernels codes and the test host code are available in the following repository: https://github.com/BioinfUD/K-mersCL. (Note: when using the kernels, the following restrictions must be considered: (1) the kernels are not an independent tool; they are designed to be part of a heterogeneous solution. (2) The size of the data set is limited according to the memory of the GPU. (3) All the reads of the data set must have equal length).

The evaluation measures and compares the execution times of the implementations of the model (OpenCL Kernels) with similar processes executed on CPU used in recognized k-mers counting tools with the aim of determining if the model does represent an alternative to parallel co-processing (many-core) for building efficient heterogeneous solutions with simultaneous CPU-GPUs execution that integrate the current methods on multi-core and the model of processing on many-core proposed in this project.

Reference tools: The two seed-based k-mers counting tools that show (in their publications) the best performance in terms of processing time and memory use were selected. For the canonical minimizer kernel (kmercl-min) the MSPKmerCounter tool was used, and for the signatures kernel (kmercl-sig) the KMC2 tool was used. MSPKmerCounter is a disk-based approach, to efficiently perform k-mer counting for large genomes using a small amount of memory. It is based on a novel technique called Minimum Substring Partitioning (MSP). In [9] it is stated that the experiment results on large real-life short reads data sets demonstrate that MSPKmerCounter can achieve better overall performance than state-of-the-art k-mer counting approaches.

KMC2 uses a novel method (signature-based) for k-mer counting, on large datasets at least twice faster than the strongest competitors (Jellyfish 2 [24], KMC 1 [25]), using about 12 GB (or less) of RAM memory.

In order to exclusively measure the time that MSPKmerCounter uses to perform the identification of super k-mers (excluding disk reading / writing times), a timer was introduced in Partition.java. KMC was modified in such way that the timer did not measure any reading / writing from/to disk. Time was measured only for the process that identifies the super k-mers by signatures; since this process can be executed sequentially several times for each thread, the times of each execution per thread were accumulated and the resulting times for each thread were averaged.

Data set: A data set consisting of 4 files generated from short reads from the sequencing of chromosome 23th of Homo sapiens was used. The files vary in the number of reads (1.5 and 9 million reads) and in their length (180 and 300 bases).

Computers: The evaluation was carried out on two computers in order to measure the performance of the model under the limitations of a desktop computer (Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz, GeForce GTX 750Ti, 16GB DDR3, SSD 1TB, Ubuntu 16.04) and under the advantages of a HPC - High Performance Computer (Intel(R) Xeon(R) CPU E5-2697 v3 @ 2.60GHz, Nvidia k80, 128GB DDR3, SSD 480GB, CentOS 7.3), which are part of the Centro de Cómputo de Alto Desempeño de la Universidad Distrital (CECAD): http://cecad.udistrital.edu.co/.

Configuration: The k-mers counter tools were configured to run serially (1 thread) and in parallel (4 threads) on CPU. For the kernels, indexed processing spaces were configured as follows: global space: two-dimensional space with a number of rows equal to the reads that make up the set and a number of columns equal to twice the number of m-mers per k-mer; local space: one-dimensional space (1 row) with a number of columns equal to those assigned to the global space. Both the kernels and the processes in the k-mers counting tools were evaluated for typical lengths of k-mers (k = 51, k = 81) and m-mers (m = 5, m = 7).

For further information on the evaluation (modifications of reference tools, data set, specific commands, among others), please refer to: https://github.com/BioinfUD/K-mersCL/blob/master/README.md.

4. RESULTS

A processing model was obtained that efficiently parallelizes the search of super k-mers (based on either minimizers or signatures seeds) on many-core architectures using two new algorithms of parallelization that maximize the operational intensity and a structure of data that substantially reduces the memory requirement for the representation of the output data (identification of super k-mers). The model was implemented through two OpenCL kernels, one for minimizer and one for signatures. Figs. 6 and 7 show the results of the evaluation made to the kernels in terms of execution times; the execution times of the kernels are represented with a bar that has two levels: the lower level corresponds exclusively to the processing time on the GPU and the upper level includes the transfer times between the host and the GPU in both directions).

The memory requirement for the search process of super k-mers in series or in parallel with few threads (on CPU) is very low considering that it is not necessary to load the entire data set into memory. The data is divided into small batches that are sequentially loaded and processed. The search for super k-mers on many-core platforms is the opposite case as it is necessary to transfer to the memory of the GPU and to process as much data as possible in a single call to the kernel due to the high cost of data transfer between the host and the GPU in both directions. Figs. 8 and 9 show the results of the evaluation; the memory requirements for the processes of the reference tools are shown as information and not with comparative purposes due to the reasons outlined above.

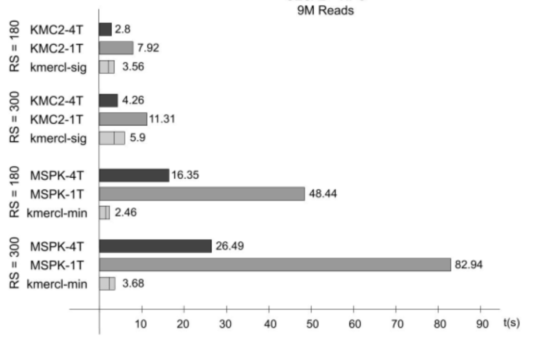

Figure 6 – Execution time. Kernels vs. references tools on the CECAD - HPC (RS: Read size; T: Threads; Kernels times: Processing | Processing + Data Transfer)

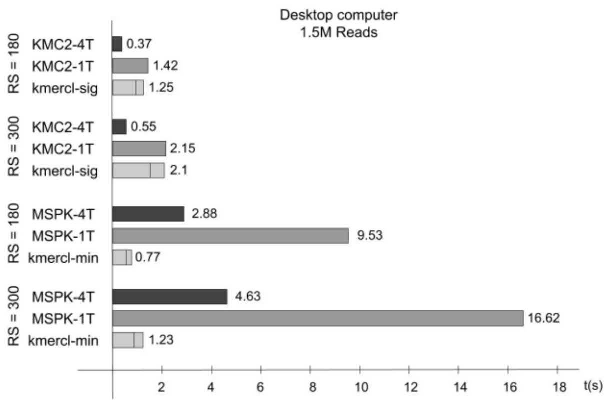

Figure 7 – Execution time. Kernels vs references tools on the desktop computer (RS: Read size; T: Threads; Kernels times: Processing | Processing + Data Transfer)

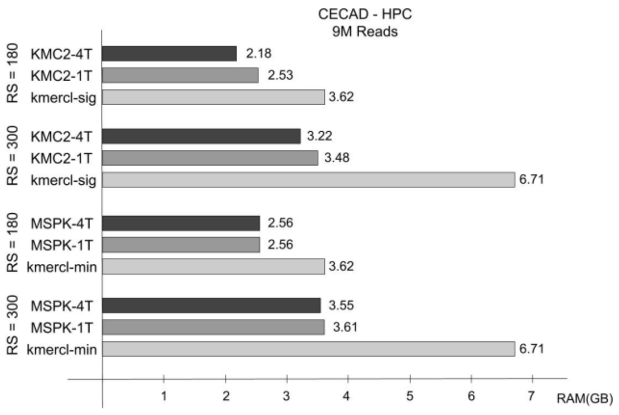

Figure 8 – Memory. Kernels and references tools on the CECAD - HPC (RS: Read size; T: Threads)

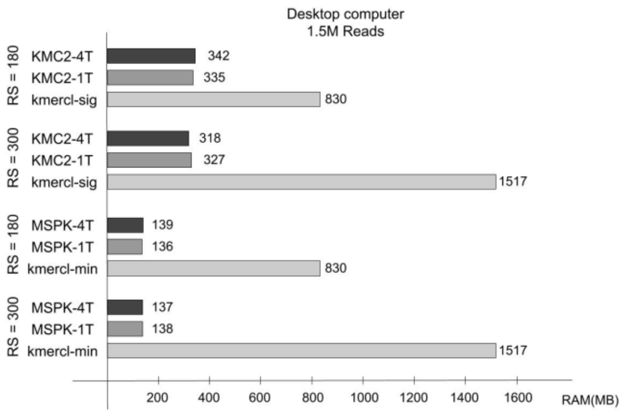

Figure 9 – Memory. Kernels and references tools on the desktop computer (RS: Read size; T: Threads)

5. DISCUSSION

The execution times of the kernels (including data transfer times) were shorter than the times of similar processes running serially (1 thread) on CPU in both reference tools and for both computational environments (for minimizer, the kernel was 19.33 times faster on average; for signature, the kernel was 1.78 times faster on average). When the reference tools run in parallel (4 threads) on CPU, the minimizer kernel is faster on both computers (6.19 times faster on average), while the signature kernel is slower (on average, the reference tool was 1.6 times faster). The relation of execution times between the kernels and the similar processes in the reference tools remained practically constant for both computing environments.

The memory requirement for both kernels depends on the input data (roughly linear relation) but not on the output data. The transient data and the output data do not influence the memory requirement; this means that efficient representation and proper use of the hierarchical memory structure of the GPU are performed. The memory requirement for both cores is predictable, so an efficient distribution of workload between the host and the GPUs is possible.

6. CONCLUSIONS

Through two new massive parallelization algorithms, focused on maximizing the operational intensity through efficient access to the hierarchical memory structure of the parallel device, and a new data structure for representing the super k-mers of a read in an indexed and compact way, it was possible to obtain a processing model that efficiently solves the search for super k-mers over many-core architectures.

The implementation of the model through two kernels OpenCL and its evaluation made it possible to demonstrate that both the execution time (including data transfer times host -- GPU -- host) and the memory requirement are adequate to use the model in the development of heterogeneous solutions of simultaneous execution (distributed work-load). The execution times of the kernels in the majority of times were lower than the times used by the current methods executed on CPU; this means that the model is a good option of simultaneous (multi-core / many-core) co-processing to accelerate the current methods executed on CPU. The memory requirement of the kernels was totally predictable and slightly high since it depends exclusively on the input data and not on the transient or output data, so the model facilitates the efficient distribution of workload between the CPU (current methods) and the GPUs (methods proposed in this paper).

7. ACKNOWLEDGEMENTS

CIDC (Centro de Investigaciones y Desarrollo Científico), CECAD (Centro de Cómputo de Alto

Desempeño), GICOGE (Grupo Internacional de Investigación en Informática, Comunicaciones y Gestión del Conocimiento) - Universidad Distrital Francisco José de Caldas.

8. FUNDING

This work has been supported by the CIDC (Centro de Investigaciones y Desarrollo Científico de la Universidad Distrital Francisco José de Caldas).

9. REFERENCES

[1] H. Li, A. Ramachandran, and D. Chen, "GPU acceleration of advanced k-mer counting for computational genomics," *Proceedings of the 2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2018, pp. 183-186.

[2] T. Richert, "Management of distributed dynamic data with algorithmic skeletons," *Parallel Computing*, 2000, pp. 375-382. https://www.worldscientific.com/doi/abs/10.1142/9781848160170_0044

[3] F. Wrede and S. Ernsting, "Simultaneous CPU–GPU execution of data parallel algorithmic skeletons," *International Journal of Parallel Programming*, vol. 46, no. 1, pp. 42–61, Apr. 2017.

[4] M. Roberts, W. Hayes, B. R. Hunt, S. M. Mount, and J. A. Yorke, "Reducing storage requirements for biological sequence comparison," *Bioinformatics*, vol. 20, no. 18, pp. 3363–3369, 2004.

[5] G. Marçais, D. Pellow, D. Bork, Y. Orenstein, R. Shamir, and C. Kingsford, "Improving the performance of minimizers and winnowing schemes," *Bioinformatics*, vol. 33, no. 14, pp. i110–i117, Dec. 2017.

[6] S. Altschul, "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs," *Nucleic Acids Research*, vol. 25, no. 17, pp. 3389–3402, Jan. 1997.

[7] N. Pérez, M. Gutierrez, and N. Vera, "Computational performance assessment of k-mer counting algorithms," *Journal of Computational Biology*, vol. 23, no. 4, pp. 248–255, 2016.

[8] M. Kokot, M. Długosz, and S. Deorowicz, "KMC 3: counting and manipulating k-mer statistics," *Bioinformatics*, vol. 33, no. 17, pp. 2759–2761, Apr. 2017.

[9] Y. Li, et al. MSPKmerCounter: a fast and memory efficient approach for k-mer counting. arXiv preprint arXiv:1505.06550, 2015.

[10] Y. Li, P. Kamousi, F. Han, S. Yang, X. Yan, and S. Suri, "Memory efficient minimum substring partitioning," *Proceedings of the VLDB Endowment*, vol. 6, no. 3, pp. 169–180, Jan. 2013.

[11] R. Chikhi, A. Limasset, and P. Medvedev, "Compacting de Bruijn graphs from sequencing data quickly and in low memory," *Bioinformatics*, vol. 32, no. 12, pp. i201–i208, 2016.

[12] C. Marchet, M. Kerbiriou, and A. Limasset, "Indexing De Bruijn graphs with minimizers," Nov. 2019. https://www.biorxiv.org/content/10.1101/546309v2

[13] J. Luo, J. Wang, W. Li, Z. Zhang, F. X. Wu, M. Li, & Y. Pan, "EPGA2: memory-efficient de novo assembler," *Bioinformatics*, vol. 31, no. 24, pp. 3988-3990, 2015.

[14] S. Deorowicz, "FQSqueezer: k-mer-based compression of sequencing data," Scientific Reports, vol. 10, 578, 2020. https://doi.org/10.1038/s41598-020-57452-6

[15] S. Deorowicz, A. Debudaj-Grabysz, and S. Grabowski, "Disk-based k-mer counting on a PC," BMC Bioinformatics, vol. 14, no. 1, 2013.

[16] M. Erbert, S. Rechner, and M. Müller-Hannemann, "Gerbil: a fast and memory-efficient k-mer counter with GPU-support," *Algorithms for Molecular Biology*, vol. 12, no. 1, 2017.

[17] Y. Ono, K. Asai, and M. Hamada, "PBSIM: PacBio reads simulator—toward accurate genome assembly," *Bioinformatics*, vol. 29, no. 1, pp. 119–121, Apr. 2012.

[18] A. Rhoads and K. F. Au, "PacBio Sequencing and Its Applications," *Genomics, Proteomics & Bioinformatics*, vol. 13, no. 5, pp. 278–289, 2015.

[19] V. Alessandrini, "Atomic types and operations," *Shared Memory Application Programming*, pp. 167–190, 2016.

[20] J. E. Stone, D. Gohara, and G. Shi, "OpenCL: A parallel programming standard for heterogeneous computing systems," *Computing in Science & Engineering*, vol. 12, no. 3, pp. 66–73, 2010.

[21] N. Vera, C. Rojas and J. Pérez, OpenCL Práctico, first ed., Editorial UD, Bogotá, 2019, 314 p.

[22] A. Klöckner, N. Pinto, Y. Lee, B. Catanzaro, P. Ivanov, and A. Fasih, "PyCUDA and PyOpenCL: A scripting-based approach to GPU run-time code generation," *Parallel Computing*, vol. 38, no. 3, pp. 157–174, 2012.

[23] A. Klöckner, N. Pinto, B. Catanzaro, Y. Lee, P. Ivanov, and A. Fasih, "GPU scripting and code

generation with PyCUDA," *GPU Computing Gems Jade Edition*, pp. 373–385, 2012.

[24] G. Marçais, C. Kingsford, "A fast, lock-free approach for efficient parallel counting of occurrences of k-mers," *Bioinformatics*, vol. 27, no 6, pp. 764-770, 2011.

[25] S. Deorowicz, A. Debudaj-Grabysz, S. Grabowsky, "Disk-based k-mer counting on a PC," *BMC Bioinformatics*, vol. 14, no. 1, p. 160, 2013.

***Nelson Enrique Vera-Parra**, Electronic Engineer from Surcolombiana University, Masters in Information Sciences and Communication from Universidad Distrital Francisco José de Caldas and PhD in Engineering from the same university. Professor at the Faculty of Engineering of the Universidad Distrital Francisco José de Caldas.*

***Danilo Alfonso López-Sarmiento**, Electronic Engineer from Pamplona University, Masters in Teleinformatic from Universidad Distrital Francisco José de Caldas and PhD in Engineering from the same university. Professor at the Faculty of Engineering of the Universidad Distrital Francisco José de Caldas.*

***Cristian Alejandro Rojas-Quintero**, Systems Engineer and Magister in Software engineering from Universidad Distrital Francisco José de Caldas. Bioinformatics developper for the agreement between the CECAD (Center for High Performance Computing of the Universidad Distrital Francisco José de Caldas) and IGUN (Institute of Genetics at the Universidad Nacional de Colombia).*