32nd European Modeling & Simulation Symposium
17th International Multidisciplinary Modeling & Simulation Multiconference

ISSN 2724-0029 ISBN 978-88-85741-44-7 © 2020 The Authors.
DOI: 10.46354/i3m.2020.emss.001

Usefulness of HIGH-LEVEL Parallel Compositions in Genomics

Mario Rossainz-López[1, *], Sarahí Zúñiga-Herrera[1], Iván Olmos Pineda[1], Ivo Pineda-Torres[1], Manuel Capel-Tuñón[2]

[1] Faculty of Computer Science, Autonomous University of Puebla, San Claudio Avenue and South 14th Street, City, San Manuel, Puebla, Puebla, 72000, México
[2]Software Engineering Department, College of Informatics and Telecommunications ETSIIT, University of Granada, Daniel Saucedo Aranda s/n, Granada 18071, Spain

*Corresponding author. Email address: mrossainzl@gmail.com

Abstract

This work shows the use of parallel objects to build High Level Parallel Compositions or HLPC and their usefulness in genomics through four case studies related to sequencing DNA chains. The first two case studies are combinatorial optimization problems: grouping fragments of DNA sequences and the parallel exhaustive search (PES) of RNA strings that help the sequence and assembly of DNAs in the construction of gnomes. The third case study shows the implementation of a Convolutional Neuronal Network as a Parallel Object Composition to solve the problem of the recognition of DNA sequences from a database with 4 types of hepatitis C virus (type 1, 2, 3 and 6). The results of this classification were obtained in terms of percentages of training precision and validation precision. The fourth and final case study shows the problem of sequence typing (STP) as a form of DNA sequence classification. It is particularized in a proposal for a parallel solution to find conserved regions of sequences that help discriminate between different types of hepatitis C virus, through the creation of a decision tree using HLPC. We show the algorithms that solves these problems using modeling and parallel simulation, their design and implementation as HLPC and the performance metrics in their parallel execution using multicores, video accelerator card and CPU-SET or processors with shared-distributed memory.

Keywords: HLPC, DNA Sequences, Parallel Objects, Structured Parallel Programming, Modeling & Simulation

1. Introduction

A computer simulation is a computation that models the behavior of some real or imagined systems over time. Computer simulations have become an important and useful part of the mathematical models of many natural science systems such as physics, electronics, astrophysics, chemistry and biology, Fujimoto (2000). Parallel Simulation refer to technologies that enable a simulation program to execute on a computing system containing multiple processors, Wilkinson, and Allen (2000). Parallel simulation then refers to the use of technologies that allow a simulation program to run in a computer system that contains several processors physically speaking, or several heavy or light processes at the programming language level. In this work we use both approaches to propose the use of parallel objects to build High Level Parallel Compositions or HLPC and their usefulness in genomics through four case studies related to sequencing DNA chains. A DNA sequence consists of an alphabet that is formed with the letters

of the four nitrogenous bases that compose it: adenine, thymine, guanine and cytosine. In the area of genomics this is important because different types of sequence tests can be used that identify, for example, infectious agents present in a blood sample taken from a patient, for diagnostic tasks. The rapid speed of sequencing attained with modern DNA sequencing technology has been instrumental in the sequencing of complete DNA sequences, or genomes of numerous types and species of life, Pareek, Smoczynski and Tretyn (2011). There are many proposals to obtain partial and complete DNA sequences quickly, obtaining speed in the calculations using current technology. A direct way to reduce the time of sequencing of DNA chains for different purposes within genomics is to parallelize the algorithms and solution techniques to the problems that arise. The reader can consult the references Peña et-al (2014), Sanjuan, Arnau and Claver (2008) and Yang et-al (2008) that relate to this work. The present research uses the structured parallel programming to propose an implementation of a library classes named high level parallel compositions or HLPC, Corradi et al (1995), Danelutto and Torquati (2014). HLPCs are based on the object-orientation paradigm to solve problems that are prone to parallelization by using a class of concurrent active objects that provide the programmer with the most common communication patterns in parallel programming: farms, pipeline and Three n-arity, Capel and Troya (1994). With them, simulations related to obtaining DNA sequences within Genomics have been solved. Four case studies are shown to demonstrate the usefulness of the HLPC, case study-1: Parallel exhaustive search of RNA strings using HLPC, case study-2: Assembly of DNA sequences using HLPC, case study-3: Recognition of DNA sequences using convolutional neuronal network with HLPC and case study-4: DNA sequence typing with decision trees using HLPC. Finally, for each case study, the performance obtained in the speed of its parallel executions and the scalability of the speedup compared to Amdahl's Law are shown, with respect to the number of CPU-SET used in the executions of each HLPC.

2. State of the art and motivation

The transformation of existing sequential applications into parallel ones for multiprocessors environments has been of great interest for decades. One alternative is to opt for parallel and concurrent programming algorithms at a high level of abstraction by using patterns of communication/interaction between processes. In Collins (2011), the effectiveness and applicability of automatic techniques has been explored. FastFlow is a framework intended to propitiate high-level, pattern-based parallel programming proposed by Aldinucci et-al (2014). The ParaPhrase project of Torquati (2015) develop frameworks and offer to users constructs, templates and parallel communication patterns between processes. Myoupo and Tchendji (2014) offers an efficient coarse parallel algorithm to solve the optimal binary search tree problem by using a binary tree as communication pattern between the processes involved. Some environments of parallel programming, as the one called SklECL, Steuwer et-al. (2011), are based on skeletons an wrappers that make up the fundamental constructs of a coordination language, defining modules that encapsulate code written in a sequential language and three classes of skeletons: control, stream parallel, and parallel data. After reviewing the literature on the research topic, we are interested to do research work that has to do with parallel applications that use predetermined communication patterns, among other component-software. At least, the following ones have currently been identified as important open problems: The lack of acceptance structured parallel programming environments of use to develop applications, the necessity to have patterns or HLPCs (High Level Parallel Composition), determination of a complete set of patterns as well as of their semantics and adoption of an object-oriented approach provide uniformity, genericity and reusability. The pattern of HLPC has deserved special interest from us.

3. HIGH-LEVEL Parallel Compositions

The basic idea here is to use classes to implement any type of parallel communication patterns between the processes of an application or distributed/parallel algorithm, thus following the object orientation. Starting from these classes, there will be objects (class instances) and the execution of any object method can be carried out through a service request. A HLPC comes from the composition of three object types:

- An object manager that controls a set of objects references, that address the collector object and several stage objects and represent the HLPC components whose parallel execution is coordinated by the object manager.
- The objects stage are objects of a specific purpose, in charge of encapsulating a client-server type interface that settles down between the manager and the slave-objects. These objects are external entities that contain the sequential algorithm that constitutes the solution of a given problem. Additionally, they provide the necessary inter-connection to implement the semantics of the communication pattern whose definition is sought.
- A collector object is an object in charge of storing the results received from the stage objects to which is connected in parallel with other objects of HLPC composition. During a service request the control flow within the stages of a HLPC depends on the implemented communication pattern. When the composition finishes its execution an instance of the collector class that is in charge of storing these results and sending them to the manager, which send the results to the environment.

Manager, collector and stages are included in the definition of a PO, Corradi et-al (1995).

3.1. The HLPC Pipeline

By using the pipeline parallel processes design technique, the problem becomes divided in a series of tasks that must be completed with a sequential dependency between each and the next one, i.e., one after another. In a pipeline each task can be executed by a process, thread or processor independently, Roosta (1999). The pipeline processes are sometimes called stages of the pipeline, Rossainz and Capel (2008). Each stage can contribute to the solution of the total problem and it can pass on the information that the following stage of the pipeline needs. Many times, this type of parallelism is seen as a form of functional decomposition. Figure 1 represents the pipeline parallel pattern of communication as a HLPC.

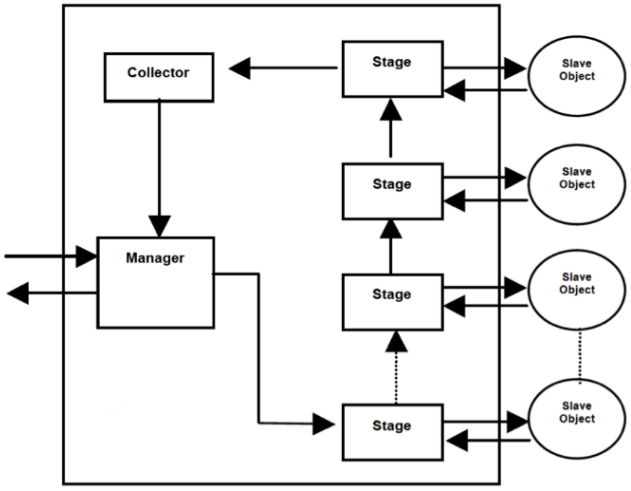


Figure 1: The HLPC of a Pipeline

3.2. The HLPC Farm

The so named farm parallel pattern of interaction is made up of a set of independent processes, called worker processes, and a process that controls them, called the process controller by Roosta (1999), Rossainz and Capel (2008). The worker processes are executed in parallel until all of them reach a common objective. The process controller oversees distributing the work and of controlling the progress of the farm until the solution of the problem is found. Figure 2 shows the pattern of the farm as HLPC.

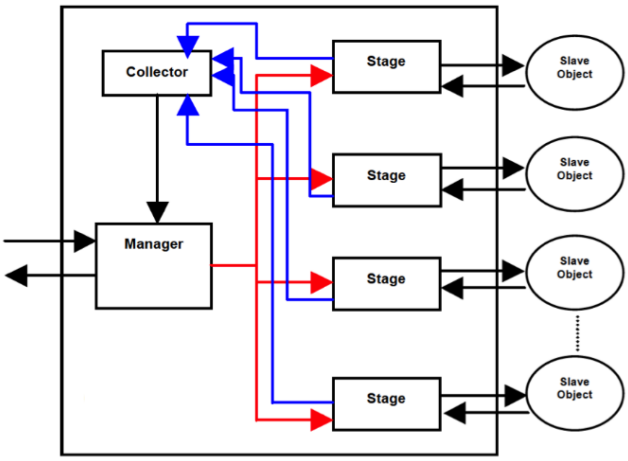


Figure 2: The HLPC of a Farm

3.3. The HLPC Tree

For simplicity the HLPC of a binary tree is shown, although it can be generalized to an arity-n tree. The nodes of the tree are represented by processors, processes or threads. The root node of the tree receives as input a complete problem that is divided into two parts. One part is sent to the left-son node, while the other is sent to the node that represents the right-son. This division process progresses recursively until the lowest atomic level of nodes in the tree is reached. After a certain time, all the leaf-nodes receive as input a subproblem given by its father-node, then they solve it and the solutions are again sent to their ancestors. Any father in the tree will obtain the partial solutions from its children and will combine them to provide only one solution that will be send to its own father node when it finishes. Finally, the root node will deliver the complete solution of the problem on finishing, Hansen (1993). Figure 3 shows the graphic representation of an arity-2 tree as an HLPC.

4. Case Study 1: Parallel exhaustive search of RNA strings

In this work, it is proposed by using the HLPC model to carry out a Parallel Exhaustive Search (PES) of RNA or DNA strings using the communication pattern called FARM. The PES was carried out in plain text files containing a representation of RNA strings of an organism with its respective name. In the HLPC Farm used, the process controller or manager performs the pre-processing of the file extracting the strings written in the FASTA format (see Pearson and Lipman (1988) for more details) to create the dictionary formed of the characteristics of the strings and the strings themselves, which is sent to each worker process or stage to perform the exhaustive search using the associated algorithms in the manager object and in the slave objects of the HLPC. This search is carried out in parallel by all the farm worker processes.

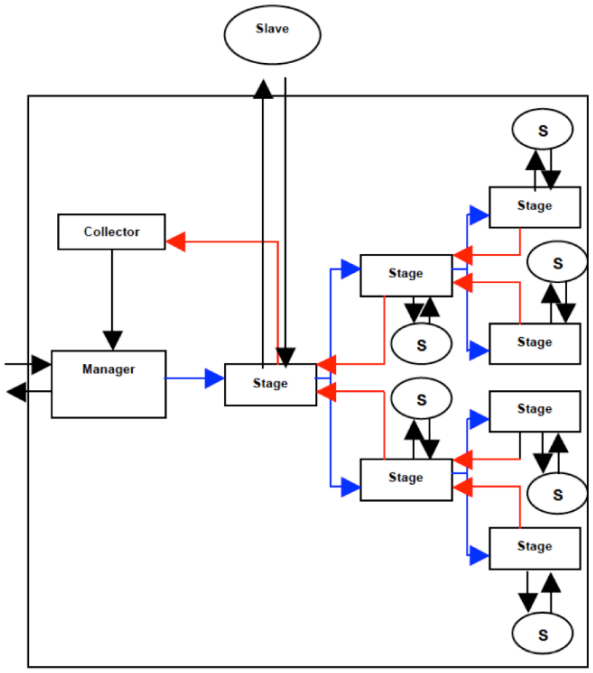


Figure 3: The HLPC of a Tree-Divide & Conquer

The HLPC Farm always guarantees a workload balance of these processes thanks to the synchronization restriction of the maximum parallelism that its components have guaranteeing the reduction in the execution times of each worker process, but also of the HLPC Farm itself. Then a new model of HLPC called HLPC ARNi is created, which is shown in Figure 4.

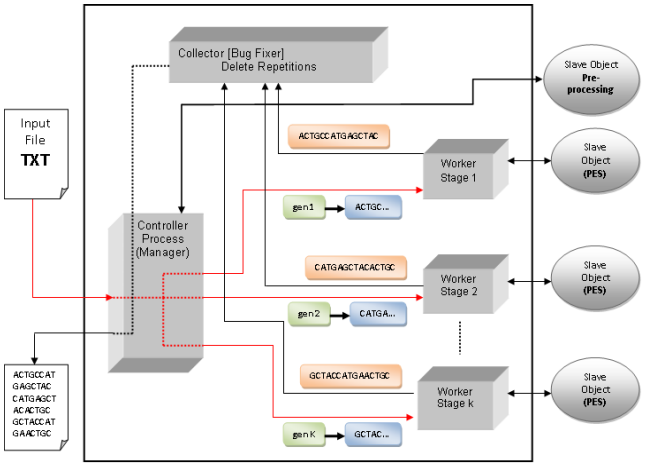
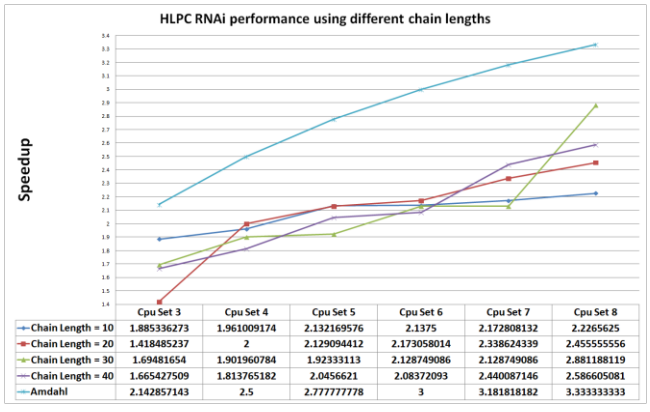


Figure 4: Model of HLPC ARNi

The pre-processing of the data is done through a text file as input to the HLPC ARNi. This file contains the name of a text string, as well as its characteristics, which is sent to the Manager of the HLPC or Farm Controller Process. The Manager has the Pre-Processing through a Slave Object, which consists of joining in a single string, all the ARNi strings that are in the TXT input file including the line text that contains the characteristics of the organism in question, then the Manager distributes to each Stage process the corresponding workload defining the limits start and end of the PES for each Stage. The load balance is made using the maximum parallelism in each Stage process (worker) of the Farm which is based on the identifier number of each worker process, Lee, et-al (2007), Levitin (2003). Subsequently the PES is performed in each stage of the HLPC ARNi executing the associated algorithm through the corresponding Slave Object and if a predefined substring is found within the ARNi string it is sent to the Collector object, which receives them in parallel from all the stage processes (workers) connected to it. The Collector bugs fixer eliminates repetitions of strings and the result is sent to the Manager who in turn sends it to an output file to the user. A simulation was designed using the HLPC ARNi with the RNAi string database located at the Pombase site (ftp://ftp.ebi.ac.uk/pub/databases/pombase/pombe/Chromosome_Dumps/fasta/). Easts are the agents of fermentation and are found on the surface of plants, Wood, et-al (2003). The experiment was carried out on a server machine with Intel Xeon processor 2630 2.40 GHz and 8 cores. Experiments were performed with different number of nodes to determine if the workload was performed correctly. To determine if the length of the strings to be searched has a direct impact on the execution time, experiments with different string lengths were performed for their search, where the execution times increase according to the length of the search string grows, but on the other hand the execution times decrease as the number of nodes (nuclei) that are used in the execution of the search increases. In Figure 5 shows the scalability of the Speedup found in the HLPC RNAi for different string lengths using 3 to 8 nodes in its execution, showing generally a good acceleration as the number of nodes increases.



	Cpu Set 3	Cpu Set 4	Cpu Set 5	Cpu Set 6	Cpu Set 7	Cpu Set 8
Chain Length = 10	1.885336273	1.961009174	2.132169576	2.1375	2.172808132	2.2265625
Chain Length = 20	1.418485237	2	2.129094412	2.173058014	2.338624339	2.455555556
Chain Length = 30	1.69481654	1.901960784	1.92333113	2.128749086	2.128749086	2.881188119
Chain Length = 40	1.665427509	1.813765182	2.0456621	2.08372093	2.440087146	2.586605081
Amdahl	2.142857143	2.5	2.777777778	3	3.181818182	3.333333333

Figure 5: Scalability of the magnitude of the Speedup found for the HLPC RNAi in exclusive nodes of 2, 3, 4, 5, 6, 7 and 8 cores

5. Case Study 2: Assembly of DNA sequences

The use of HLPCs for grouping DNA sequence fragments from the parallelization of a clustering algorithm to evaluate a set of fragments are made, which have a high probability of being aligned in an assembly task, Masoudi-Nejad et-al (2013), DanishAli

and Farooqui (2013). The algorithm finds the splices between the fragments using the Myers algorithm and links them in a graph. Then an in-depth search is done in the graph to form the groups and send them as a result. The assembly of DNA strings is proposed as a combinatorial optimization problem and is classified as NP-hard and is based on the paradigm divide-and-conquer using a structure type farm, so that the computational cost of finding the sequence alignments and its splice is substantially reduced with respect to its sequential version. The number of processes required to process the fragments of DNA sequences of a specific genome such as that of a virus or bacteria is determined by the splice of the strings found by the sequential solution algorithm, which looks in parallel for overlaps in the remaining fragments. Two sub-strings of each fragment are taken for comparison with other fragments; and thus, splices are located and associated with the processes. A splice graph is then generated that shows the relationship between pairs of nodes, as well as the lack of communication among others. The set of nodes of the graph that are inter-related are grouped together within a worker process pattern farm. Each set of related nodes in the graph are independent and represent the grouping of fragments found. In Figure 6 is shown the representation of HLPC for grouping DNA sequence fragments.

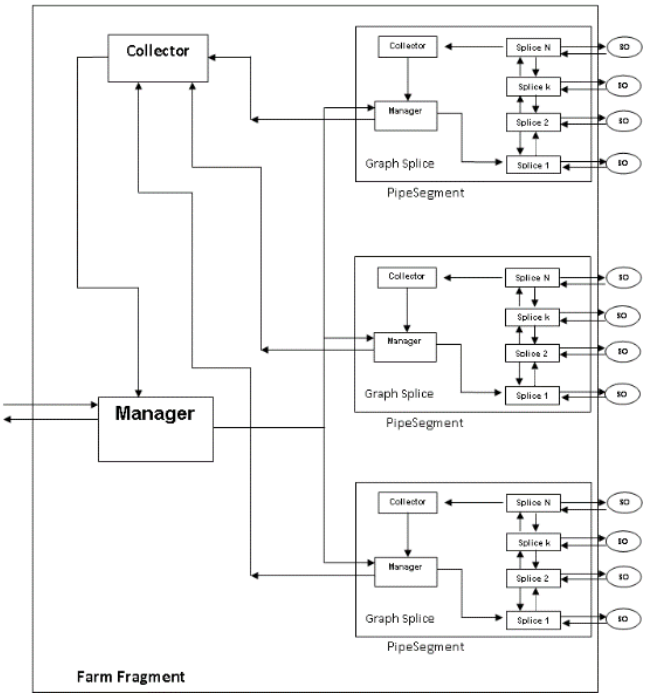


Figure 6: The HLPC GraphADN

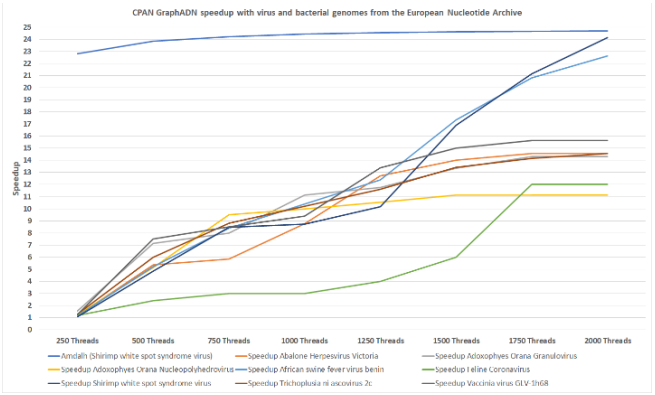


Figure 7: HLPC GraphADN Speedup found with virus and bacterial genomes from the European Nucleotide Archive

The new HLPC named HLPC GraphADN is structured as a FARM of n-fragments of DNA sequences and each worker process is itself a two directions-communication pipeline HLPC formed by m-stages where each stage of HLPC Pipe represents a splice sequence of DNA strings connected with both, the previous stage as the next stage. The collector object receives the number of formed groups and the elements that belong to each of the formed groups. With the latter information collected, an in-depth search is performed to locate these items and obtain the sequence groups formed by the sequential algorithm assigned to each of the HLPC's slave objects with this result, the user can use an assembly of DNA sequences to try to complete a particular genome or to finish an incomplete sequence of DNA strings of some animal or plant type species. An simulation was designed by using the HLPC GraphADN with genomes of viruses and bacteria available on the web whose data were obtained from European Nucleotide Archive, is shown in Figure 7. The plot shows the number of processes deployed for the calculation of eight genomes in an experiment conducted on a computer Intel Core i8 processor and using a video accelerator card with 1,664 CUDA cores.

6. Case Study 3: Recognition of DNA sequences using convolutional neuronal network (CNN)

A CNN is an algorithm for machine learning in which a model learns to perform classification tasks directly from images, videos or sounds, Calvo (2015). The parallelization of a convoluted neuronal network under the HLPC model is shown. The HLPC Pipeline is adapted to a convoluted neuronal network model to the transfer learning technique; which allows its execution in parallel computers or computers with GPUs. Convolutional networks have characteristics of neural networks such as activation functions or fully connected layers, but also introduce two concepts: the convolutional layer and the grouping or sampling layer. The architectures of convolutional networks are built by stacking these elements, that is why according to the computational and memory use issues of a neural

network for image processing, Marturet and Alferez (2018), it is useful and appropriate to represent it through an HLPC pipeline. For the training of a convolutional neuronal network, the transfer of learning by extraction of deep descriptors was used as a way of training and validating the neural network on the set of images of the specific problem to be solved, Marcelo, et-al (2000). In this way we obtain the HLPC Pipeline-CNN that is shown in the figure 8, and that will help to solve the case study that is shown below.

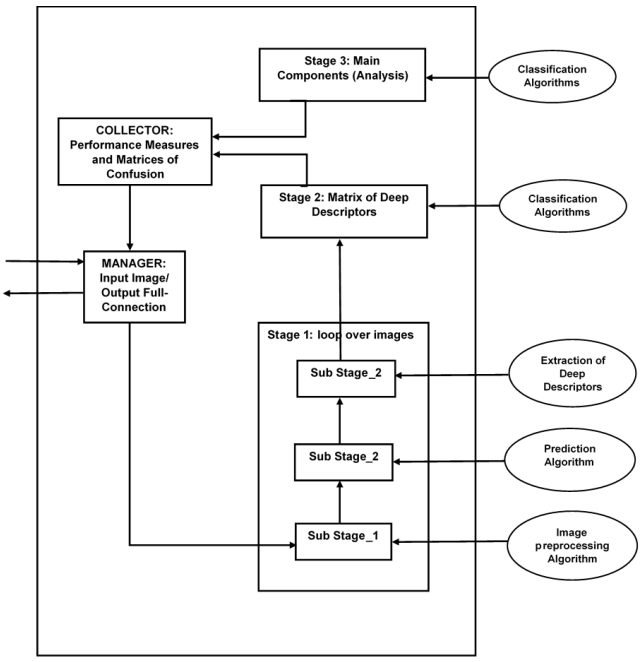


Figure 8: HLPC Pipeline-CNN

The idea is to convert the DNA sequences to graphic representations to train the HLPC Pipeline-CNN. The DNA sequences are represented by letters: A-Adenine, G-Guanine, C-Cytosine and T-Thymine, however, a CNN is not made to process information with this format, so a graphic representation of the sequences was designed. We used 1847 DNA sequences from a database with 4 types of hepatitis C virus (type 1, 2, 3 and 6) taken from the repository available on the ViPR page (https://www.viprbrc.org/brc/home.spg?decorator=vipr) and a set of DNA sequences from the Molecular database that has 3190 sequences, available on the UCI page (https://archive.ics.uci.edu/ml/index.php).

The computer equipment used for the training of the HLPC Pipeline-CNN was a parallel computer with 64 processors of which only 32 were exclusive for the tests of this work, 8 GB of main memory with a distributed shared memory architecture and high-speed buses. Regarding classification results for the HLPC Pipeline-CNN trained with the database of the four types of Hepatitis C virus, a precision of 95% was obtained with 145 images tested and at the end of step 4000 the precision training was 94.5% and precision validation 95%. The graphs in Figure 9 show the performance analysis of the HLCP Pipeline-CNN from 1000 training steps to 4000 training steps respectively.

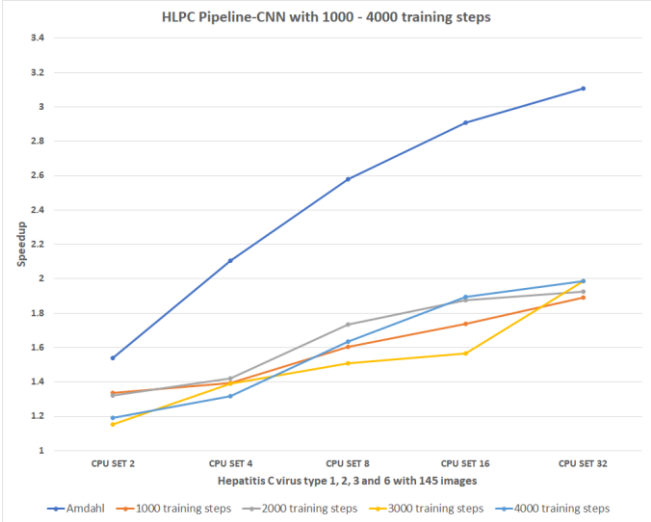


Figure 9: Speedup scalability found for HLPC

Pipeline-CNN of Precision training and precision validation with 1000-4000 training steps for Hepatitis C virus type 1,2,3 and 6. In this graph the speedup of the precision training and precision validation of HLPC Pipeline-CNN with classes of Hepatitis C virus type 1, 2, 3 and 6 is illustrated. In her, the speedup shows an acceleration to be incorporating more CPU-SET, always below the law of Amdahl. The execution times in each training vary.

7. Case Study 4: DNA sequence typing with decision trees

The problem of sequence typing (STP) is shown as a form of DNA sequence classification. It is particularized in a proposal for a parallel solution of finding conserved regions of sequences that help discriminate between the different types of hepatitis C virus through the creation of a tree of decision using High Level Parallel Compositions (HLPC) and that the researchers carry out the design of primers and diagnostic tests of polymerase chain reactions (PCR) when they try to detect different types of viruses, in less time. A decision tree is a structure formed by a set of nodes, leaves and branches that represents a prediction model whose objective is inductive learning from observations and logical constructions, Barrientos, Cruz and Acosta (2009). The root node of the tree is the attribute from which the classification process begins, the internal nodes correspond to each of the questions about the attribute of the problem. Each possible response is represented by a child node. The branches that leave each of these nodes are labeled with the possible attribute values. The leaf nodes correspond to a decision, which coincides with one of the class variables of the problem to be solved (Barrientos, Cruz and Acosta, 2009). The definition of the CPAN-DesicionTree (see Figure 10) is shown as an integrated

part of a solution proposal (entropy, information gain and decision tree) in the problem to be solved: In a set of sequences or instances Gw of DNA we want to locate those attributes Ai that provide more information and are considered the best attributes to solve the classification problem of the seven Cy classes of the hepatitis C virus that currently exist. As an example of simulation, the data in Table 1 is used to show that it is possible to classify DNA sequences using the concepts of entropy and information gain. In this case, attribute $A3$ with greater mayor (Information Gain) is selected, this attribute quickly discriminates classes $C1$ and $C4$. When calculating again IG of all the attributes, it is obtained that both $A1$ and $A2$ allow to discriminate classes $C2$ and $C3$. This is obtained by generating a decision tree where each vertex has a maximum of 4 possible values. This structure is created using the CPAN-DesicionTree (Figure 10).

Table 1. Representation of S set of instances Gw, where each instance belongs to a class Cy.

Cy	Gw	A1	A2	A3	A4	A5
C1	G1	A	T	T	A	T
C1	G2	A	T	T	C	T
C_2	G_3	A	G	C	A	C
C_2	G_4	A	G	C	G	C
C_2	G_5	A	G	C	A	T
C_2	G_6	A	G	C	G	T
C_3	G_7	G	T	C	T	C
C_3	G_8	G	T	C	T	C
C_3	G_9	G	T	C	A	G
C_3	G_{10}	G	T	C	T	G
C_4	G_{11}	A	G	A	A	C
C_4	G_{12}	A	G	A	G	C

The CPAN-DesicionTree receives through its Manager process the database or repository with the DNA sequence instances of the Hepatitis-C virus. The Manager process sends the information to the first Stage process that represents the decision tree root and that has a slave object associated with the mathematical algorithms and models of Shannon's entropy and information gain. Then there is an attribute considered as "best attribute" that is solving the classification problem or a branch that generates more nodes and can obtain the best attributes that are sent to the Collector process who receives them to form the best solution set attributes of the classification problem. The set of best attributes is sent to the Manager process, which in turn sends them to the user as the result of the process. The execution of the DesicionTree CPAN processes is carried out in parallel, with the restriction, synchronization and process communication policies, whose details can be found in Collins (2011), Ernsting and Kuchen (2012). Figure 11 shows the scalability of the Speedup found in the CPAN-DesicionTree from 3 to 8 cores, obtaining a good acceleration.

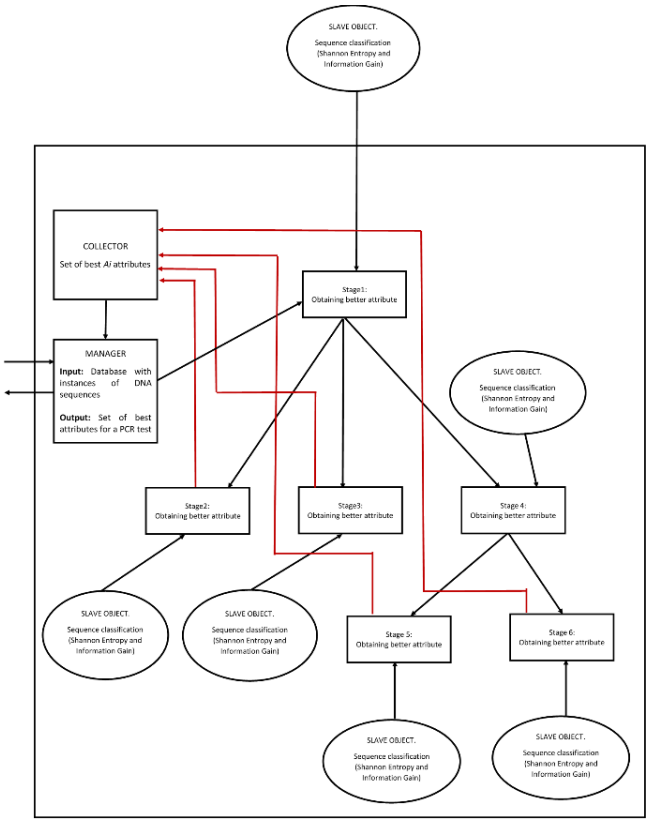


Figure 10: Decision tree represented as the CPAN-DesicionTree to solve the problem of classification in DNA sequences

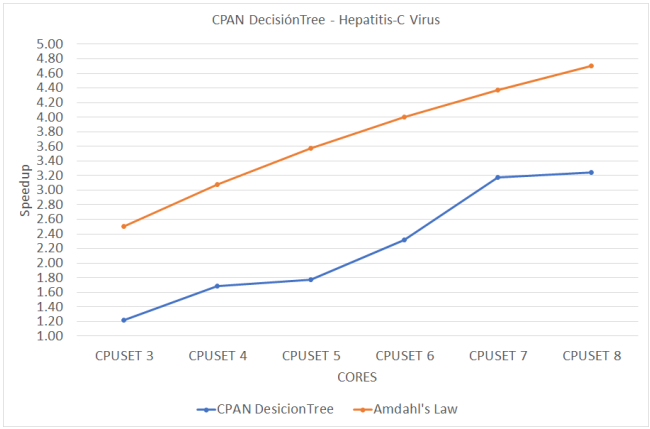


Figure 11: Speedup scalability found for CPAN-DesicionTree in the problem of classification of Hepatitis-C virus DNA sequences from the example in Table 1.

8. Conclusions

We discuss the design, implementation and simulation of parallel applications based on the HLPC. In a way we discuss the implementation of HLPCs pipeline, farm and Tree as patterns of communication/interaction between processes, which can even be used by inexperienced parallel application programmers to obtain efficient code by only programming the sequential parts of their applications. We have

presented four case studies: the parallel exhaustive search of RNAi strings through the new HLPC RNAi constructed, the parallel calculation of the DNA sequences for 8 genomes, the implementation of a Convolutional Neuronal Network as a Parallel Object Composition to solve the problem of the recognition of DNA sequences from a database with 4 types of hepatitis C virus and the solution to the problem of sequence typing (STP) as a form of DNA sequence classification to find conserved regions of sequences that help discriminate between different types of hepatitis C virus, through the creation of a decision tree using HLPC. In all cases of study, the efficiency and speedup scalability of the HLPCs in the solution of the problems has been shown.

References

Aldinucci, M., Danelutto, M., Kilpatrick, P. and Torquati, M. 2014. FastFlow: high-level and efficient streaming on multi-core. Programming Multi-core and Many-core Computing Systems, Wiley.

Barrientos Martínez R.E., Cruz Ramírez N., Acosta Meza H.G., et-al, 2009. Árboles de Decisión como herramienta en el diagnóstico médico, Revista Médica de la Universidad Veracruzana, Volumen 9, Número 2, Veracruz, México.

Calvo D. 2015. Red Neuronal Convolucional (CNN). Data Scientist. http://www.diegocalvo.es/red-neuronal-convolucional/

Capel, M., & Troya, J. M., 1994. An Object-Based Tool and Methodological Approach for Distributed

Collins A.J. 2011. Automatically Optimising Parallel Skeletons. MSc thesis in Computer Science, School of Informatics University of Edinburgh, UK.

Corradi A, Leonardo L, Zambonelli F., 1995. Experiences toward an Object-Oriented Approach to Structured Parallel Programming. DEIS technical report no. DEIS-LIA-95-007.

Danelutto M. and Torquati M, 2014. Loop parallelism: a new skeleton perspective on data parallel patterns. Parallel Distributed and Network-based Processing, Torino, Italy.

DanishAli S. and Farooqui 2013. Approximate Multiple Pattern String Matching using Bit Parallelism. International Journal of Computer Applications, Volume 74, No.19, pp. 47–51.

Ernsting S. and Kuchen H. 2012. Algorithmic skeletons for multi-core, multi-GPU systems and clusters, Int. J. of High-Performance Computing and Networking, Vol. 7, No. 2, pp.129–138.

Fujimoto R.M. 2000. Parallel and Distributed Simulation Systems. Wiley, Hoboken (2000).

Hansen B., 1993. Model Programs for Computational Science: A programming methodology for multicomputers. Concurrency (Chichester, England), 5(5).

Lee R.C.T., Tseng S.S., Chang R.C., Tsai Y.T., 2007. Introducción al diseño y análisis de algoritmos, un enfoque estratégico. Mc Graw Hill.

Levitin A., 2003. The Design of Analysis of Algorithms. Wesley.

Marcelo A., Apolloni J., Kavka C., et-al 2000. Entrenamiento de Redes Neuronales. Universidad Nacional de San Luís. WICC 2000. Argentina.

Marturet R., Alferez E.S., 2018. Evaluación de Redes Neuronales Convolucionales para la clasificación de imágenes histológicas de cancer color rectar mediante transferencia de aprendizaje. Master en Bioinformática y Bioestadística. Universitat Oberta de Catalunya. España.

Masoudi-Nejad, A., Narimani, Z. and Hosseinkhan, N. 2013. Next Generation Sequencing and Sequence Assembly. SpringerBriefs in Systems Biology.

Myoupo, J.F. and Tchendji, V.K. (2014). Parallel dynamic programming for solving the optimal search binary tree problem on CGM, International Journal of High Performance Computing and Networking, Vol. 7, No. 4, pp.269–280.

Pareek C., Smoczynski R. and Tretyn A. 2011. Sequencing technologies and genome sequencing, Journal of Applied Genetics, Vol. 25, No. 4, pp.41–3435.

Pearson W.R., Lipman D.J., 1988. Improved tools for biological sequence comparison. In Proceedings of the National Academy of Sciences of the United States of America 85, http://fasta.bioch.virginia.edu/fasta_www2/fasta_list2.shtml

Peña A.J., Claver J.M., Sanjuan A., Arnau V. (2014). Análisis Paralelo de Secuencias ADN mediante el uso de GPU y CUDA, ResearchGate. https://www.researchgate.net/publication/228857228.

Rossainz, M., 2005. Una Metodología de Programación Basada en Composiciones Paralelas de Alto Nivel (HLPCs). Universidad de Granada, PhD dissertation, 02/25/2005.

Rossainz, M., Capel M., 2008. A Parallel Programming Methodology using Communication Patterns named CPANS or Composition of Parallel Object. 20TH European Modeling & Simulation Symposium.Campora S. Giovanni. Italy.

Rossainz-López Mario, Capel-Tuñón Manuel, Pineda-Torres Ivo, Olmos-Pineda Ivan, Olvera-López Arturo, 2018. Use of Parallel Patterns of Communication between Processes for search of Sequences DNA and RNAi Strings. Research in Computing Science: Applications of Language & Knowledge Engineering. Volume 148, Number 3,

ISSN: 1870-4069. México.

Roosta, S., 1999. Parallel Processing and Parallel Algorithms. In Theory and Computation. Springer.

Sanjuan A., Arnau V., Claver J.M. 2008. Análisis Paralelo de Secuencias ADN sobre computadores con multiples cores. Actas de las XIX Jornadas de Paralelismo, Castellón, España.

Steuwer M., Kegel P. and Gorlatch S. 2011. SkelCL a portable skeleton library for high-level GPU programming. Proceedings of the 16th IEEE Workshop on High-Level Parallel Programmin Models and Supportive Environments, May, Anchorage, AK, USA.

Torquati, M., Aldinucci, M. and Danelutto, M. (2015) FastFlow Testimonials, Computer Science Department, University of Pisa, Italy.

Wilkinson, B., Allen, M. 2000. Parallel Programming. Techniques and Applications Using Networked Workstations and Parallel Computers, Prentice Hall.

Wood V., Gwilliam R., Rajandream M.A., et al. 2003. The genome sequence of Schizosaccharomyces pombe. Nature.

Yang X.Y., Ripoll A., Marin I., Luque E. 2008. Genomic-scale analysis of DNA Words of Arbitrary Length by Parallel Computation. NIC Series, Vol. 33, 623–630.