DOI:https://doi.org/10.46354/i3m.2019.emss.001

REPRESENTATION OF A CONVOLUTIONAL NEURONAL NETWORK AS A HIGH LEVEL PARALEL COMPOSITION APPLIED TO THE RECOGNITION OF DNA SEQUENCES

M. Rossainz-López[a], S. Zúñiga-Herrera[a], M. Capel-Tuñón[b], I. Pineda-Torres[a]

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

[a]rossainz@cs.buap.mx, [a]sarahi.zuhe@gmail.com, [b]manuelcapel@ugr.es, [a]ipineda@cs.buap.mx

ABSTRACT

This work proposes the use of Structured Parallel Programming using the process communication pattern called Pipeline in its version of High-Level Parallel Composition (HLPC) to implement a process composition that represents a convolutional neuronal network or CNN and that is used to solve a specific problem of DNA sequences. The HLPC Pipeline-CNN is then shown, which represents the implementation of a convolutional neural network making use of the three types of parallel objects that make up an HLPC: A manager object, one or more stage objects and a collector object. The manager object represents the HLPC itself and makes an encapsulated abstraction out of it that hides the internal structure, the stage objects are objects of a specific purpose, in charge of encapsulating an client-server type interface that settles down between the manager and the slave-objects and the collector object that is an object in charge of storing the results received from the stage objects to which is connected. To show the usefulness and performance of the HLPC Pipeline-CNN implemented, it was used in 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, as well as performance results in terms of speedup from 1000 to 4000 training steps with 2, 4, 8, 16 and 32 exclusive processors in one parallel machine of up to 64 processors with shared-distributed memory.

Keywords: High Level Parallel Compositions, HLPC, Convolutional Neuronal Network, CNN, Deep Learning Transfer, DNA Sequences, Parallel Objects, Structured Parallel Programming.

1. INTRODUCTION

The convolutional neural networks (CNN) are like multichannel neural networks, its main advantage is that each part of the network is trained to perform a task, this significantly reduces the number of hidden layers, so training is faster (Calvo 2015). The convolutional neural networks are very powerful for everything that has to do with the image analysis. However, its use is not only restricted to image analysis, it can also be applied to the speech recognition, or to the classification of sentences, with the necessary transformations regarding the type of the input data (Calvo 2015, Vizcayab2018). A convolutional neuronal network is a multilayer network consisting of alternating convolutional and reduction layers, like a pipeline architecture (Calvo 2015). In the convolution, operations of products and sums are carried out between the starting layer and the n filters that a characteristic map (matrix) generates. The extracted characteristics correspond to each possible location of the filter in the original image. The advantage is that the same filter (neuron) serves to extract the same characteristic in any part of the input, with this it is possible to reduce the number of connections and the number of parameters to train in comparison with a multilayer network of total connection (Calvo 2015, Vizcayab2018). In the reduction the number of parameters is reduced by staying with the most common characteristics. The last layer of this network is a sorting layer that will have as many neurons as the number of classes to predict. However, one of the disadvantages of neural networks is the large amount of time needed for training. A direct way to reduce this time is to parallelize the learning algorithms. However, algorithms cannot always be parallelized in a simple way, and in addition, the amount of communication between processors or processes means that most parallel versions of these algorithms can only be executed properly in parallel computers. That is why, in this work, we propose a parallelization of a convolutional neural network under the model of High-Level Parallel Compositions or HLPC as an original and useful proposal to obtain a good performance in the use of a CNN within a concrete problem. HLPC are parallel patterns defined and logically structured that, once identified in terms of their components and of their communication, can be adopted in the practice and be available as high-level abstractions in user applications within an OO-programming environment (Brinch Hansen 1993). The process interconnection structures of parallel execution patterns such as trees can be built using HLPCs, within the work environment of POs that is the one used to detail the structure of a HLPC implementation (Corradi and Leonardi 1991). A structured approach to parallel programming is based on the use of communication/interaction patterns which are predefined structures of user's application processes (Wilkinson and Allen 1999). In such a situation, the

structured parallelism approach provides the interaction-pattern abstraction and describes applications through HLPCs, which are able to implement the pattern mentioned already (Darlington 1993). The encapsulation of a HLPC should follow the modularity principle and it should provide a base to obtain an effective reusability of the parallel behavior to be implemented. When there is the possibility of attaining this, a generic parallel pattern is built, which in its turn provides a possible implementation of the interaction structure between processes of the application, independently of the functionality of these processes. Several significant and reusable parallel patterns of interconnection can be identified in multiple applications and parallel algorithms (Roosta and Séller 1999) which has resulted in a wide library of communication patterns between concurrent processes such as HLPCs whose details are found in (Rossainz and Capel 2008; Rossainz and Capel 2014). In the present work we propose the implementation of HLPC Pipeline-CNN which represents a convolutional neural network using deep learning transfer, as a learning strategy for the neural network and it was used for the recognition of 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) The set of DNA sequences used is the Molecular database (Splice-junction Gene Sequences) Data Set that has 3190 sequences, available on the UCI page (https://archive.ics.uci.edu/ml/index.php), with three classes of sequences: limit exon-intron, limit intronexon and any. For the use of the DNA sequences a representation method was designed where each nitrogenous base is represented in gray scale to form an image. The generated images were used to train the convolutional neuronal network HLCP Pipeline-CNN. The results are shown both of the classification carried out by the HLPC Pipeline-CNN in terms of training precision and validation precision, as well as of parallel performance in its execution obtaining measurements of the law of Amdahl and speedup in a parallel computer with 32 exclusive CPU-SET.

2. DEFINITION OF HIGH-LEVEL PARALLEL COMPOSITIONS (HLPC)

Using an OO-programming environment, the basic idea is implementing any type of parallel communication patterns between the processes of an application or distributed/parallel algorithm. A HLPC comes from the composition of a set three object types: the manager object, the stage objects and the collector object. The object manager (Figure 1) represents the HLPC itself and makes an encapsulated abstraction out of it that hides the internal structure (Rossainz 2005). The object manager controls a set of objects references, which address the object collector 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. The collector object can see 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. Manager, collector and stages are included in the definition of a PO (Corradi, Leonardo and Zambonelli 1995). POs are active objects, which have intrinsic execution capability. Applications that deploy the PO pattern can exploit the inter-object parallelism as much as the intra-object parallelism (Bacci, Danelutto, Pelagatti, and Vaneschi 1999; Danelutto and Torquati 2014). A PO-instance object has a similar structure to that of an object in C++, and additionally defines a scheduling policy that specifies the way in which one or more operations carried out by the instance synchronize (Danelutto and Torquati 2014). The communication modes used are: The synchronous communication, the asynchronous communication and the asynchronous future (Birrell 1989; Lavander and Kafura 1995). The Synchronization policies are expressed in terms of restrictions; for instance, mutual exclusion in reader/writer processes or the maximum parallelism allowed for writer processes (Andrews 2000).

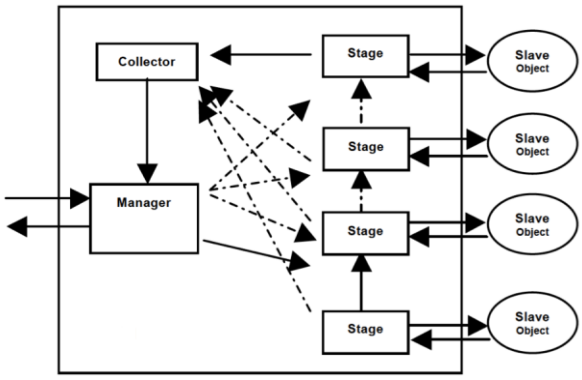

Figure 1: Internal structure of HLPC

The Figure 1 shows the pattern HLPC without defining any explicit parallel communication pattern. The box that includes the components, represents the encapsulated HLPC, internal boxes represent compound objects (collector, manager and objects stages), as long as the circles are the objects slaves associated to the stages. The continuous lines within the HLPC suppose that at least a connection should exist between the manager and some of the component stages. Same thing happens between the stages and the collector. The dotted lines mean more than one connection among components of the HLPC.

2.1. Construction of Communication Patterns between Processes as HLPC

Currently there is a class library that provides the programmer with the three communication patterns between processes most commonly used as HLPC: The

process farm, the process pipeline and the process tree (initially binary process trees), (Liwu 2002). Figure 2, Figure 3 and Figure 4 show the farm, pipeline and tree models as High-Level Parallel Compositions or HLPC. These models are abstract. The programmer must adapt them to the problem he is trying to solve by making use of the properties of the paradigm of object orientation such as inheritance or polymorphism. The implementation details can be found in (Rossainz and Capel 2008). The structure of the library is shown in the class diagram of Figure 5. The details are in (Rossainz and Capel 2014).

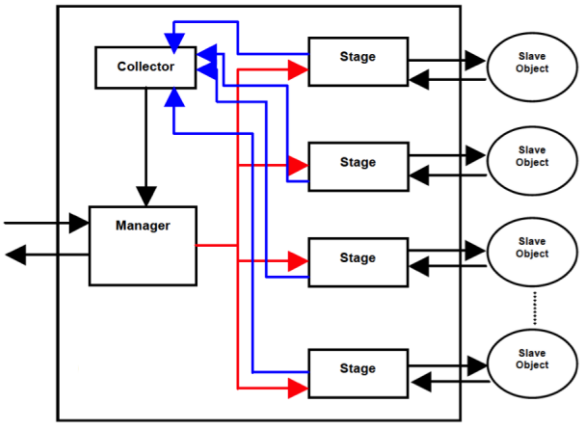

Figure 2: The HLPC of a Farm

With the proposed library it is possible to build concrete HLPCs. To build an HLPC, first it should have made clear the parallel behavior that the user application needs to implement, so that the HLPC becomes this pattern itself. Once identified the parallel behavior, the second step consists of elaborating a graph of its representation. This practice is also good for illustrating the general characteristics of the desired system and will allow us to define its representation with HLPCs later, by following the pattern proposed. When the model of a HLPC has already been made clear, it defines a specific parallel pattern; let's say, for example, a tree, or some other mentioned pattern, and then the following step will be to do its syntactic definition and specify its semantics (Liwu 2002).

Finally, the syntactic definition prior to any programmed HLPC is transformed into the most appropriate programming environment, with the objective of producing its parallel implementation.
The HLPC models of figures 2, 3 and 4 have been used to adapt them and generate farms, pipeline and particular trees, of problems that have been solved with this proposal such as: ordering, search and optimization problems, NP-Complete problems like that of the Traveling Agent, simulation problems such as the movement and attraction of particles in space and more recently with problems that have to do with finding DNA sequences in the construction of GNOMAS.

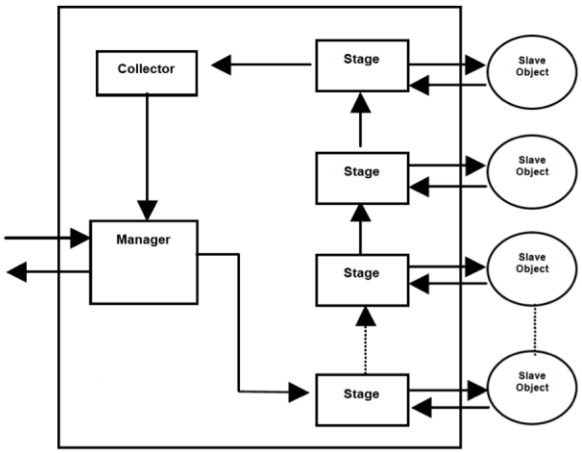

Figure 3: The HLPC of a Pipeline

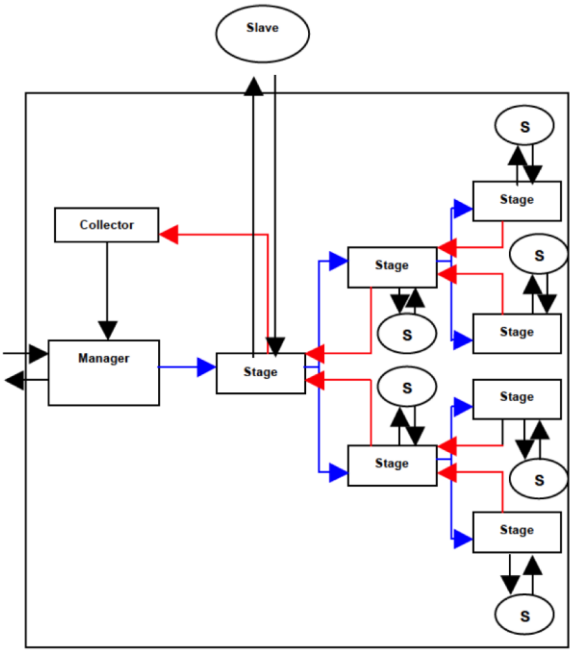

Figure 4: The HLPC of a Tree-Divide & Conquer

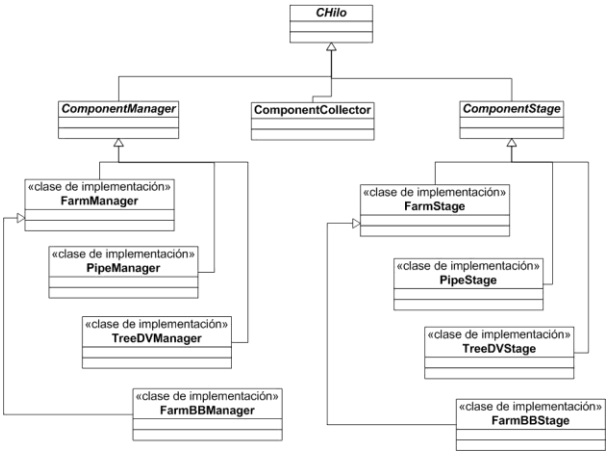

Figure 5: Structure of the HLPC class library

3. CONVOLUTIONAL NEURAL NETWORK (CNN)

In recent years, the field of machine learning has progressed enormously in addressing problems of classification, identification and pattern recognition. In particular, it has been found that a type of model called convolutional neural network or CNN achieves reasonable performance in hardware visual recognition tasks, equaling or exceeding human performance in some domains (Salzberg, Searls and Kasif 1998). A CNN is an algorithm for machine learning in which a model learns to perform classification tasks directly from images, videos or sounds. CNNs are especially useful for locating patterns in images in order to recognize objects, faces and scenes. They learn directly from the image data, using patterns to classify the images and eliminate the need for a manual extraction of features. For a CNN to learn, deep learning models are used. The most common is Inseption-V3 that is designed for the Visual Recognition challenge, this is a standard task in artificial vision, where the models try to classify complete images in 1000 ImageNet classes. This model is available in TensorFlow, which is a tool for machine learning. TensorFlow is designed primarily for deep neural network models (Mathworks. 2018). Modern models of image recognition have millions of parameters; training them from scratch requires a lot of tagged training data and a lot of computing power. Therefore, one of the disadvantages of neural networks is the large amount of time needed for training. A direct way to reduce this time is to parallelize the learning algorithms. However, the algorithms cannot always be parallelized in a simple way, and, the amount of communication between the processes, makes that most of the parallel versions of these algorithms can only be executed in parallel computers (Marcelo, Apolloni, Kavka 2000). Transfer learning is a quick technique that takes a piece from a model that has already been trained in a related task and reuses it in a new model, Figure 6 (taken from) shows an example of a CNN, the filters are applied to each training image with different resolutions, and the output of each convolved image is used as input for the next layer generating a communication pattern of pipeline type and that can be parallelized (Mathworks 2018, Marcelo, Apolloni, Kavka 2000).

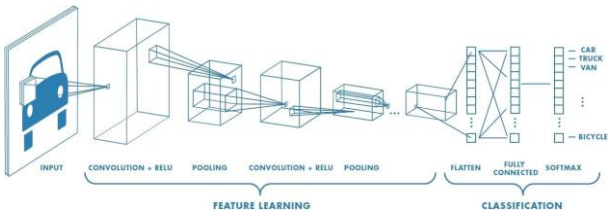

Figure 6: Example of a Convolutional Neural Network (taken from Mathworks 2018)

The transfer learning technique is effective for many applications, works with moderate amounts of training data (thousands, not millions of tagged images) and can be executed sequentially in thirty minutes (Salzberg, Searls and Kasif 1998). In this work, 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.

4. REPRESENTATION OF A CNN AS A HLPC USING DEEP LEARNING TRANSFER

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. In this way we obtain the HLPC Pipeline-CNN that is shown in the figure 7, and that will help to solve the case study that is shown in following sections in this article.

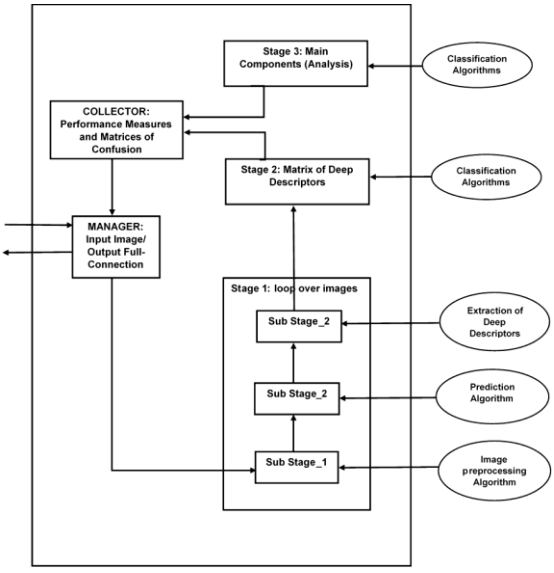

Figure 7: HLPC Pipeline-CNN

In the HLPC Pipeline-CNN of figure 7, in the first convolutional layer, the neurons that make up the CNN connect to a portion of the input image provided by the user in the HLPC manager object and not to the whole of it. When several convolutional layers are concatenated, to a part of the output of a layer, certain neurons of the next layer are connected, but not all of them that make up the second layer. This is carried out in the first stage of the HLPC Pipeline-CNN (loop over

images). After several concatenated convolutional layers, details are obtained regarding the characteristics of the image, for example shapes or colors. In each convolutional layer, feature maps are stacked. A feature map is a layer where all its neurons use the same filter and share characteristic parameters. In each convolutional layer as many filters are applied as feature maps are stacked in it. The transfer of learning in the HLPC Pipeline-CNN is produced by describing deep descriptors in the sub-stage2 of the model. This adjustment occurs through training and validation of the set of images of the specific problem that is solved. The work is completed in stage 2 and stage 3 of the HLPC Pipeline-CNN, executing the classification algorithms associated with the slave objects, to generate the matrix of deep descriptors and the analysis of main components, obtaining as results within the collector's object, model performance measures, confusion matrices and the network's classification layer (see figure 6 and figure 7). The internal execution of the parallel objects of the HLPC including the manager, the collector and the stages, as well as the inter-object parallelism of the HLPC, make the solution of the specific problem that is solved obtain a better performance than its sequential counterpart, when working with a large number of images of the order of hundreds of thousands under a hardware architecture that is also parallel.

5. RECOGNITION OF DNA SEQUENCES USING HLPC PIPELINE-CNN

The processes of gene prediction are those that, within the area of computational biology, are used for the algorithmic identification of pieces of genomic DNA sequences (Christos Ouzounis 2012), and that are biologically functional. The identification of genes is an important area to understand the genome of a species once it has been sequenced (Salzberg, Searls and Kasif 1998). DNA is composed of four molecules called nucleotides or nitrogenous bases: adenine, thymine, guanine and cytosine (Panduro 2009). A DNA sequence is composed of an alphabet that contains the letters of the four nitrogenous bases (figure 8).

GTAGTCATGTTGAAAAACTTACGAGTAAATTACGTTGTCGA
GGGCGTGCAAGTAGCGCAACCCGTGACAAGCGCAAATTCG
GAAGTAT**ACGCCAA**TCTACCGCCTCCCGTACCCGCGGAGAC
GTATCAAACCGACGAAGATTACGAGGAAGATGACGGAGG
GTGGGC

Figure 8: A DNA sequence

A DNA sequence can define the characteristics of a living organism, containing all the genetic information in units of inheritance called genes. Splicing junctions are points in a DNA sequence in which "useless" DNA is removed during the process of creating proteins in higher organisms. The problem then is to recognize with a DNA sequence, the boundaries between the exons (the parts of the DNA sequence that are retained after splicing) and the introns (the parts of the DNA sequence that are cut). This problem consists of two subtasks: recognition of exon / intron limits (called “EI” or donor) and recognition of intron / exon boundaries (“IE” sites or acceptor), (Noordewier, Towell and Shavlik 1991). Both tasks are complicated since there is no standard sequence to recognize introns and exons, which is why it is interesting to design tools that help us identify and classify them.

To improve the representation of a DNA chain, sequences are used that can be transformed into representation with numerical or alphabetic values: A (adenine), T (thymine), G (guanine) and C (cytosine), (Genís, Blanco and Guigó 2000), as shows figure 8. However, the representation of large amounts of information as DNA sequences do not make their mathematical analysis easy, this creates the need to find new ways of representing information. It is presented as a case study to generate images from DNA sequences to be analyzed by deep learning techniques, using as a convolutional neuronal network the proposal of the HLPC Pipeline-CNN shown in this research and thus be able to classify images.

The idea is to convert the DNA sequences to graphic representations to train the HLPC Pipeline-CNN. Remember that CNN are used for the recognition of patterns and classification of images. 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.

5.1. Case Study: DNA sequences of the Hepatitis-C virus

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 (Splice-junction Gene Sequences) Data Set that has 3190 sequences, available on the UCI page (https://archive.ics.uci.edu/ml/index.php), with three classes of sequences: limit exon-intron, limit intron-exon and none. The methodology used was the following:

1. A grayscale color was assigned to each of the letters of the DNA sequence (see table 1), which goes from a value of 0 = black to a value of 1 = white, so that the intermediate colors are tones of gray to show a better contrast.
2. The image representing the DNA sequences was created: A matrix of dimension 60 X 60 was used, where the value 60 coincides with the number of nitrogenous bases of all the sequences of the database.

Table 1. Grayscale of nitrogenous bases

Nitrogen base	*Value of gray*
A	0
C	0.3
G	0.7
T	1

Each sequence was placed in the first row and copied in the rest of the rows until it was 60 in total (see figure 9). The result is an image with bars in the grayscale like the one shown in Figure 10, each of the images obtained is specific to each instance of the database shown in Figure 2. In total, 3190 images were obtained.

```
1  CCAGCTGCATCACAGGAGGCCAGCGAGCAGGTCTGTTCCAAGGGCCTTCGAGCCAGTCTG
2  CCAGCTGCATCACAGGAGGCCAGCGAGCAGGTCTGTTCCAAGGGCCTTCGAGCCAGTCTG
3  CCAGCTGCATCACAGGAGGCCAGCGAGCAGGTCTGTTCCAAGGGCCTTCGAGCCAGTCTG
.
.
.
60 CCAGCTGCATCACAGGAGGCCAGCGAGCAGGTCTGTTCCAAGGGCCTTCGAGCCAGTCTG
```

Figure 9: DNA sequences to be encoded

Figure 10: Image of an instance of DNA sequences

3. With the representative images of each sequence, the HLPC Pipeline-CNN was trained. This model of HLPC is based on the CNN InceptionV3 model with deep learning transfer to categorize the recognition of three classes of DNA sequences: recognition of exon/intron limits (EI sites), recognition of intron/exon boundaries (IE sites) and recognition of none of the previous two (N).
4. With the TensorFlow software library, a classification model was constructed and placed as a slave object within the HLPC Pipeline-CNN to be parallelized (see figure 7). This was achieved by categorizing the recognition of a database with four classes of DNA sequences: Hepatitis C virus type 1, 2, 3 and 6 and the recognition of another database with three types of limits: EI, IE and N.
5. With the HLPC Pipeline-CNN the last layers of the networks were trained with instances obtained from the databases, both networks were trained in 4000 steps. First the HLPC Pipeline-CNN was trained to classify the 4 types of Hepatitis virus, then the training was done with 2 classes: IE and IE and finally with all the classes of the database: IE, IE and N to compare the results of the last two neurons.
6. Finally, classification and performance analysis results of the proposed HLPC model were obtained.

6. RESULTS AND PERFORMANCE

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% (see table 2). When using the HLPC Pipeline-CNN with the classes EI and IE, an evaluation precision of 80.8% was obtained with 177 test images and at the end of step 4000 the precision training was 82% and the precision validation was 75% (see table 3). The results of the training of the HLPC Pipeline-CNN where the three classes of the database were used show a precision of evaluation of 57.5% with 301 images and at the end of step 4000 the precision training was 69% and the precision validation 56% (see table 4).

Table 2. Precision training and precision validation of HLPC Pipeline-CNN with classes of Hepatitis C virus type 1, 2, 3 and 6.

Training steps (145 images tested)				
	1000 steps	*2000 steps*	*3000 steps*	*4000 steps*
Precision Training	92%	95%	96%	94.5%
Precision Validation	91%	92%	95%	96%

Table 3. Precision training and precision validation of HLPC Pipeline-CNN with IE and EI classes.

Training steps (177 images tested)				
	1000 steps	*2000 steps*	*3000 steps*	*4000 steps*
Precision Training	77%	80%	81.7	82%
Precision Validation	73%	74.7%	75%	75%

Table 4. Precision training and precision validation of HLPC Pipeline-CNN IE, EI and N classes.

Training steps (301 images tested)				
	1000 steps	*2000 steps*	*3000 steps*	*4000 steps*
Precision Training	57%	60%	64%	69%
Precision Validation	52%	56%	55%	56%

Regarding the performance of HLPC Pipeline-CNN for the case study that has been shown, the aim is to show that the performances obtained are "good" based on the model of the HLPC. The graphs in figures 11, 12, 13 and 14 show the performance analysis of the HLCP Pipeline-CNN from 1000 training steps to 4000 training steps respectively. In these graphs 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, IE and EI classes and IE, EI and N classes is illustrated. In all of them, the speedup shows an acceleration to be incorporating more CPU-SET, always below the law of Amdahl. The execution times in each training vary: For the case of 1000 training steps, of an average sequential execution time of 24 minutes we obtained a decrease with 32 CPU-SET of an average of 11.3 minutes. For the case of 2000 training steps, from an average sequential execution time of 28 minutes we obtained a decrease with 32 CPU-SET of an average of 17 minutes. In the case of 3000 training steps, we obtained an average sequential execution time of 33 minutes while the parallel execution with 32 CPU-SET was an average of 14.8 minutes. Finally, for the case of 4000 training steps, the average sequential execution time was 40 minutes and the parallel execution with 32 CPU-SET decreased it by an average of 20.1 minutes.

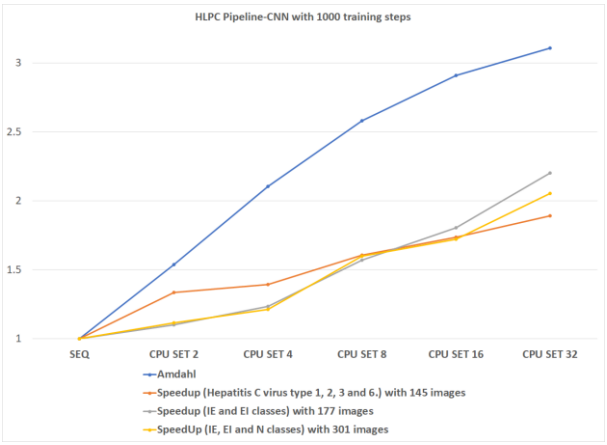

Figure 11: Speedup scalability found for HLPC Pipeline-CNN of Precision training and precision validation with 1000 training steps.

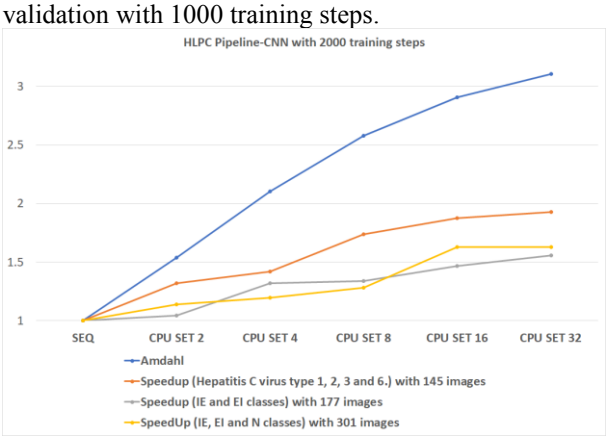

Figure 12: Speedup scalability found for HLPC Pipeline-CNN of Precision training and precision validation with 2000 training steps.

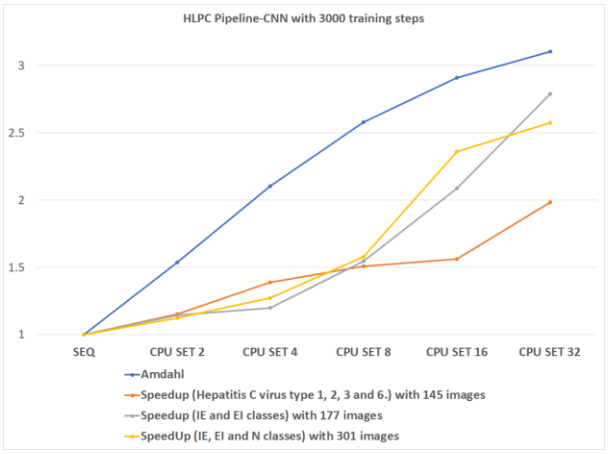

Figure 13: Speedup scalability found for HLPC Pipeline-CNN of Precision training and precision validation with 3000 training steps.

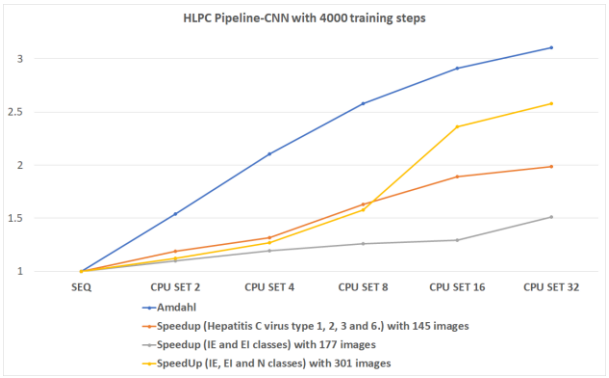

Figure 14: Speedup scalability found for HLPC Pipeline-CNN of Precision training and precision validation with 4000 training steps.

7. CONCLUSIONS

We discuss the implementation of HLPC Pipeline-CNN as generic and reusable communication/interaction patterns between processes which implements a Convolutional Neural Network (CNN) with deep learning transfer, using a pipeline as associated communication pattern. This HLPC can even be used by inexperienced parallel application programmers to obtain efficient code by only programming the **sequential** parts of their applications (slave objects of the model of figure 7). The HLPC Pipeline-CNN was used to be trained in the recognition of DNA sequences through graphic representations and be able to obtain a classification of the different types of hepatitis C virus (type 1, 2, 3 and 6). The results obtained from the HLPC Pipeline-CNN trained with the Hepatitis C virus database suggest that the automatic learning methodology used in this work is suitable for the classification of images generated from DNA sequences. Good percentages of evaluation precision, training precision and validation precision are shown.

The transfer of learning is good when there are few images available to train the HLPC Pipeline-CNN and it

allows to reach acceptable results in most cases (This can be seen in tables 2, 3 and 4), although the results can be improved. On the other hand, the parallel execution of the HLPC Pipeline-CNN shows a good performance comparing its acceleration with respect to its sequential execution. We have also obtained good performance in their executions and speedup scalability compared to Amdahl's law on the number of processors used in training (see figures 11, 12, 13 and 14).

REFERENCES

Andrews G.R., 2000. Foundations of Multithreaded, Parallel, and Distributed Programming, *Addison-Wesley*

Bacci, Danelutto, Pelagatti, Vaneschi, 1999. SklE: A Heterogeneous Environment for HPC Applications. *Parallel Computing 25*.

Birrell A., 1989. An Introduction to Programming with Threads, *Digital Equipment Corporation, Systems Research Center*, Palo Alto California, USA.

Brinch Hansen, 1993. Model Programs for Computational Science: A programming methodology for multicomputers, *Concurrency: Practice and Experience*, Volume 5, Number 5.

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

Corradi A., Leonardi L., 1991. PO Constraints as tools to synchronize active objects. *Journal Object Oriented Programming* 10, pp. 42-53.

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.

Christos Ouzounis 2012. Rise and demise of bioinformatics? promise and progress. PLoS computational biology, 8(4):e1002487.

Danelutto M. and Torquati M, 2014. Loop parallelism: a new skeleton perspective on data parallel patterns, *in Proc. of Intl. Euromicro PDP*: Parallel Distributed and Network-based Processing, Torino, Italy.

Darlington et al., 1993, Parallel Programming Using Skeleton Functions. *Proceedings PARLE'93*, Munich (D).

Genís P, Blanco P. and Guigó R., (2000). Geneid in drosophila. Genome research, 10(4):511–515.

Lavander G.R., Kafura D.G. 1995. A Polimorphic Future and First-class Function Type for Concurrent Object-Oriented Programming. *Journal of Object-Oriented Systems*.

Liwu Li, 2002. Java Data Structures and Programming. *Springer Verlag*. Germany. ISBN: 3-540-63763X.

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 Convulcionales para la clasificación de imágenes histológicas de cancer colorrectar mediante transferencia de aprendizaje. Master en Bioinformática y Bioestadística. Universitat Oberta de Catalunya. España.

Mathworks. 2018. Deep learning. https://la.mathworks.com/solutions/deep-learning/convolutional-neural-network.html

Noordewier M, Towell G, and Shavlik Jude, (1991). Training knowledge-based neural networks to recognize genes in DNA sequences. In Advances in neural information processing systems, pages 530–536.

Panduro A, 2009. Biología molecular en la clínica. McGraw-Hill Interamericana.

Roosta, Séller, 1999. Parallel Processing and Parallel Algorithms. *Theory and Computation. Springer*.

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 M., Capel M., 2014. Approach class library of high level parallel compositions to implements communication patterns using structured parallel programming. *26TH European Modeling & Simulation* Symposium. Campora Bordeaux, France.

Salzberg SL, Searls DB, and Kasif S., 1998. Computational gene prediction using neural networks and similarity search. Computational Methods in Molecular Biology, pp.32-109.

Vizcaya R., (2018). Deep Learning para la detección de peatones y vehículos sobre FPGA. Disertación de Master. Universidad Autónoma del Estado de México.

Wilkinson B., Allen M., 1999. Parallel Programming Techniques and Applications Using Networked Workstations and Parallel Computers. *Prentice-Hall*. USA.