USE OF CPANS FOR GROUPING DNA SEQUENCE FRAGMENTS IN THE CONSTRUCTION OF GNOMES

M. Rossainz-López[(a)], Manuel I. Capel[(b)], R. Hernández-Munive[(a)], I. Olmos-Pineda[(a)] , J. A. Olvera-López[(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, [(b)]manuelcapel@ugr.es, [(a)] azhu116@gmail.com, [(a)]iolmos@cs.buap.mx, [(a)]aolvera@cs.buap.mx

ABSTRACT
This paper proposes the representation through the model of the High Level Parallel Compositions or CPANs, of the communication / interaction patterns denominated Pipeline and Farm, and their usefulness in combinatorial optimization problems within the classification of NP-hard problems such as grouping fragments of DNA sequences and assembly of these fragments to construct gnome; through a Structured Parallel Programming approach based on the concept of Parallel Objects. The Pipeline and Farm models are shown as CPANs under the Object Orientation paradigm and with them it is proposed the creation of a new CPAN that combines and uses the previous ones to solve the problem of obtaining and assembling Strings of DNA. Each CPAN proposal contains a set of predefined synchronization constraints between processes (maximum parallelism, mutual exclusion and synchronization of producer-consumer type), as well as the use of synchronous, asynchronous and asynchronous future modes of communication. We show the algorithm that solves the assembly of DNA sequences, their design and implementation as CPAN and the performance metrics in their parallel execution using multicores.

Keywords: CPAN, High Level Parallel Compositions, Parallel Structured Programming, Parallel Objects, Pipeline, Farm, DNA, Contigs, Genome.

1. INTRODUCTION

Currently within the parallel programming one of the open problems of major interest is the lack of acceptance structured parallel programming environments of use to develop applications. Structured parallelism is a type of parallel programming based on communication/interaction patterns (pipelines, farms, trees, etc.) that are predefined among the processes of a user application. Patterns also encapsulate parallel parts of the application, in such way that the user will only program the sequential code of the application. Many proposals of environments exist for the development of applications and structured parallel programs (Bacci, Danelutto, Pelagatti and Vaneschi 1999; Darlington 1993), but until now only a very limited circle of expert programmers use them. In the literature there are several proposals and all agree on the importance of determining a complete set of patterns and try to define a semantics for them (De Simone 1997). At moment, in HPC, a great interest exists in structured-parallel environments research, as the ones previously mentioned. The trend is the use of object-oriented programming approaches. It has been shown that defining parallel objects for the development of new methodological proposals, models and parallel programming communication patterns has generated good results (Corradi and Leonardi 1991; Corradi and Zambonelli 1995). HLPCs or CPANs 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 most common parallel execution patterns, such as pipelines and farms can be built using CPANs, within the work environment of POs that is the one used to detail the structure of a CPAN implementation. With them, problems like the assembly of DNA strings proposed in this paper can be solved. Finding the solution to these types of problems has become indispensable in research in biology and in many fields such as medical diagnosis, biotechnology, forensic biology, virology, applied biology and bioinformatics among others. The problem of the assembly of DNA strings enters the so-called NP-Hard, because it is a problem of combinatorial optimization in which diverse heuristics and met heuristics have been proposed to assemble sequences of DNA strings and to provide essential information to understand the species and their mechanisms of life including the human species. This work shows the implementation of a grouping algorithm that evaluates a set of DNA sequence fragments as a CPAN. The CPAN represents a Farm where worker processes are themselves Pipeline CPANs. The algorithm determines subgroups of fragments by DNA sequences matching found, which have a high probability of being aligned in an assembly task. Each worker process of CPAN Farm works in parallel with the other worker processes that are generated with a group of fragments of DNA sequences that are internally constructed as graphs represented through the CPAN Pipeline and through an in-depth search the new groups of DNA sequences are generated, which must be processed by some assembly technique to form the contigs of a genome that has been sequenced covering

most of its structure but missing a fragment to be completed. Finally the design of an experiment is shown through the use of the new CPAN generated called Cpan GraphADN, with genomes of viruses and bacteria available on the web. The pseudo random synthetic readings created to form contigs are shown and the execution performance of this proposal is obtained for eight genomes with an Intel Core i8 processor, a video accelerator card with 1664 CUDA cores and a clock frequency of 1178 MHz.

2. HIGH LEVEL PARALLEL COMPOSITIONS (CPAN)

HLPCs or CPANs 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 (Rossainz 2005, Rossainz and Capel 2008). The process interconnection structures of most common parallel execution patterns, such as pipelines, farms and trees can be built using CPANs, within the work environment of POs that is the one used to detail the structure of a CPAN implementation, for details see (Rossainz and Capel 2012). A CPAN comes from the composition of a set three object types: an object manager that represents the CPAN itself and makes an encapsulated abstraction out of it that hides the internal structure. The object manager controls a set of objects references, which address the object collector and several stage objects and represent the CPAN components whose parallel execution is coordinated by the object manager (figure 2). 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 do not actively participate in the composition of the CPAN, but are considered 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 which definition is sought. In other words, each stage should act a node of the graph representing the pattern that operates in parallel with the other nodes. Depending on the particular pattern that the implemented CPAN follows, any stage of it can be directly connected to the manager and/or to the other component stages. In collector object we 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 CPAN composition. That is to say, during a service request the control flow within the stages of a CPAN depends on the implemented communication pattern. When the composition finishes its execution, the result does not return to the manager directly, but rather to an instance of the collector class that is in charge of storing these results and sending them to the manager, which will finally send the results to the environment, which in its turn sends them to a collector object as soon as they arrive, without being necessary to wait for all the results that are being obtained (Rossainz and Capel 2012). If we observe the scheme as a black box, the graphic diagram of a CPAN representation would be the one that is shown in Figure 1.

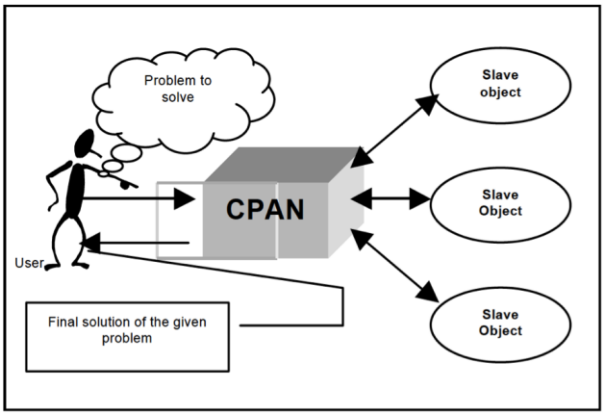

Figure 1: Graphical representation of a CPAN as black-box

In summary, a CPAN is composed of an object manager that represents the CPAN itself, some stage objects and an object of the class Collector, for each petition that should be managed within the CPAN. Also, for each stage, a slave object will be in charge of implementing the necessary functionalities to solve the sequential version of the problem being solved (Figure 2).

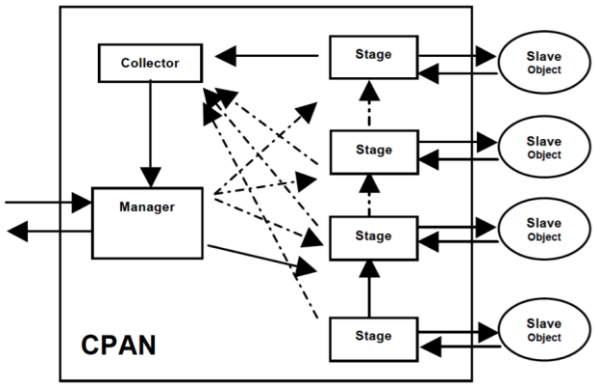

Figure 2: Internal structure of CPAN. Composition of its components

Figure 2 shows the pattern CPAN in general, without defining any explicit parallel communication pattern. The box that includes the components, represents the encapsulated CPAN, 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 CPAN 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 CPAN. For details CPAN model, see (Rossainz and Capel 2014).Manager, collector and stages are included in the definition of a Parallel Object (PO), (Corradi 1991). Parallel Objects are active

objects, which is equivalent to say that these objects have intrinsic execution capability (Corradi 1991). Applications that deploy the PO pattern can exploit the inter-object parallelism as much as the internal or intra-object parallelism. A PO-instance object has a similar structure to that of an object in Smalltalk, and additionally defines a scheduling politics, previously determined that specifies the way in which one or more operations carried out by the instance synchronize (Danelutto and Orlando 1995; Corradi 1991). Synchronization policies are expressed in terms of restrictions; for instance, mutual exclusion in reader/writer processes or the maximum parallelism allowed for writer processes. Parallel objects support multiple inheritance in the CPAN model. Parallel objects define 3 communication modes: The synchronous communication mode stops the client activity until it receives the answer of its request from the active server object (Andrews 2000), The asynchronous communication does not delay the client activity. The client simply sends the request to the active object server and its execution continues afterwards (Andrews 2000) and the asynchronous future will delay client activity when the method's result is reached in the client's code to evaluate an expression. For details see (Lavander and Kafura 1995).

3. THE CPAN PIPELINE

It represents the aforementioned pipeline technique of parallel processing as a HLPC or CPAN, applicable to a wide range of problems that are partially sequential intrinsically. The CPAN Pipe guarantees the parallelization of sequential code using the pattern PipeLine.

3.1. The technique of the Pipeline

Using the technique of the Pipeline, the idea is to divide the problem in a series of tasks that have to be completed, one after another, see figure 3. In a pipeline each task can be executed by a process, thread or processor for separate (De Simone 1997; Wilkinson and Allen 1999).

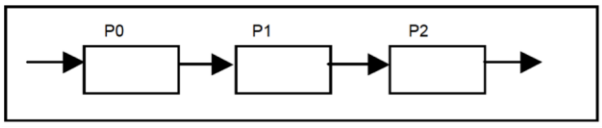

Figure 3: Pipeline

The processes of the pipeline are sometimes called stages of the pipeline (Roosta 1999). Each stage can contribute to the solution of the total problem and it can pass the information that is necessary to the following stage of the pipeline. This type of parallelism is seen many times as a form of functional decomposition. The problem is divided in separate functions that can be executed individually, but with this technique, the functions are executed in succession (Barry and Allen 1999). The technique of parallel processing pipeline is then presented as a High Level Parallel Composition applicable to solving a range of problems that are partially sequential in nature, so that the Pipe CPAN guarantees code parallelization of sequential algorithm using the pattern Pipeline.

3.2. Representation of the Pipeline as a CPAN

The Figure 4 represents the parallel pattern of communication Pipeline as a CPAN. The details of the implementation can be consulted in (Rossainz, Capel and Domínguez 2015).

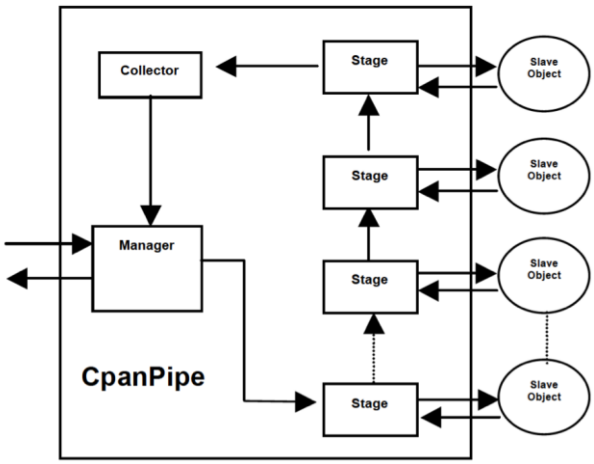

Figure 4: The CPAN of a Pipeline

Once the objects are created and properly connected according to the parallel pattern Pipeline, then you have a CPAN for a specific type of parallel pattern, and can be resolved after the allocation of objects associated with slave stages.

4. THE CPAN FARM

The technique of the parallel processing of the farm as a HLPC or CPAN is shown here. 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 (Roosta 1999) and (Rossainz and Capel 2008). The worker processes are executed in parallel until all of them reach a common objective. The process controller is in charge of distributing the work and of controlling the progress of the farm until the solution of the problem is found (Barry and Allen 1999). Figure 5 shows the pattern of the farm.

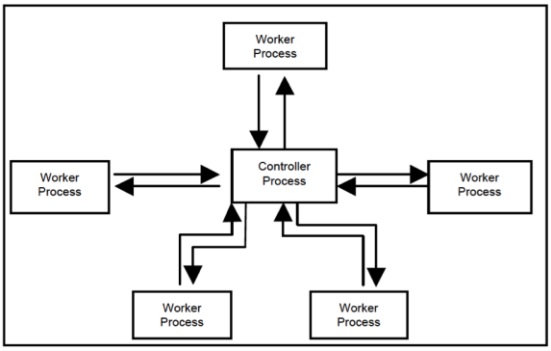

Figure 5: Farm with a controller and five workers

ISBN 978-88-97999-85-0; Affenzeller, Bruzzone, Jiménez, Longo and Piera Eds.

4.1. Representation of the Farm as a CPAN

The representation of parallel pattern farm as a CPAN is shown in Figure 6. The details of the implementation can be consulted in (Rossainz and Capel 2012).

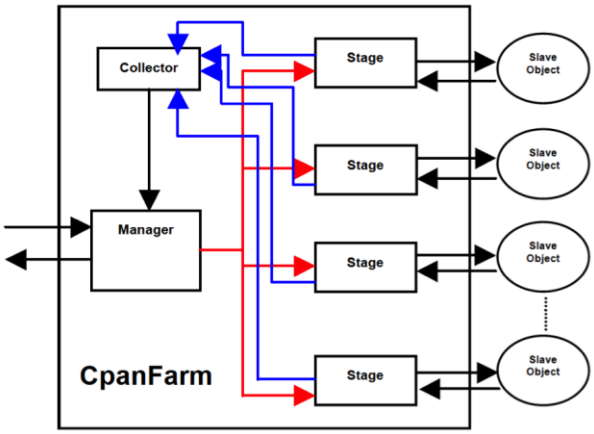

Figure 6: The CPAN of a farm

5. REPRESENTATION OF THE GENOME IN AN ORGANISM

The hereditary information of living beings is stored in a molecule of deoxyribonucleic acid called DNA. This molecule consists of two strands of nucleotides that form a structure, similar to a twisted ladder, called a double helix, see Figure 7 (Dias 2011).

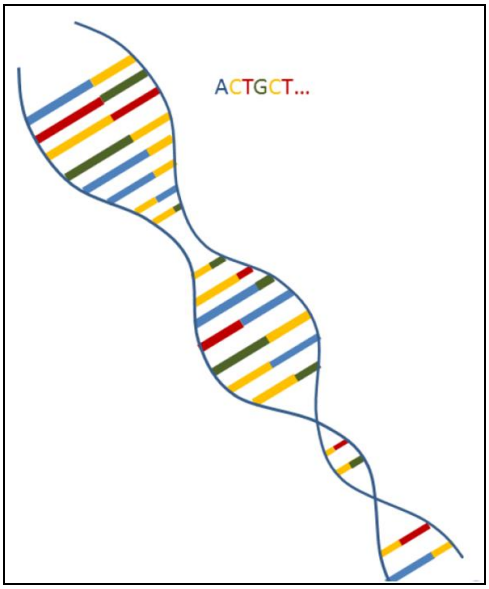

Figure 7: DNA structure, double helix

Each strand of DNA is composed of several nucleotides that are molecules formed by a nitrogenous base, a sugar that contains five molecules of carbon and a phosphoric acid. There are four types of nitrogen bases: adenine (A), guanine (G), cytosine (C) and thymine (T). In the double helix the nitrogenous bases are paired through bridges of hydrogen, giving the circumstance that the A always joins the T and the G to the C (Dias 2011). This is called base pairs (bp).

Chromosomes are filament-like structures that are inside the nucleus of a cell and contain the genetic material of a species, see Figure 8 taken from (Samiksha 2016). Each organism has a certain number of chromosomes per cell. Humans, for example, have 46 chromosomes.

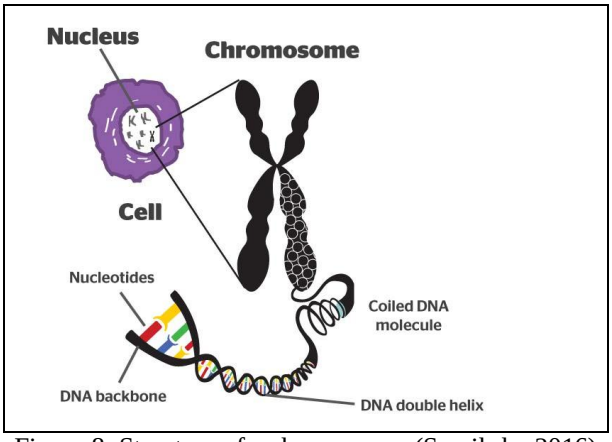

Figure 8: Structure of a chromosome (Samiksha 2016)

Along the DNA strand there are sequences of nitrogenous bases containing genetic information. These sequences are called genes and are responsible for telling cells how, when and where to produce all the necessary structures for life. All cells in the same organism have the same genetic information (Días 2011).

Genome is therefore the complete set of genetic information contained in the chromosome. Both the genome and the DNA sequences belonging to it are measured by counting their number of base pairs (bp). For very long sequences, as for a complete genome, kbp, Mbp and Gbp are used. According to the species, the genome may be composed of hundreds of thousands to billions of base pairs. However, the length of the genome is not related to the degree of complexity of a species. There are similar organisms that may differ in the size of their genomes and more complex organisms that may have smaller genomes than other simpler ones. The number of genes in a genome also has no relation to the size of the genome. But, the number of chromosomes can influence the number of base pairs of a given species. With all this we have that, the exact representation of the genome is formed of 3 parts: the assembly of DNA sequences, correction of error and the formation of contigs and scaffolding.

5.1. The sequence assembly of DNA

One of the initial problems in genome study is the measurement of the similarity of DNA sequences within the same genome but also within different genomes of equal species or within genomes of different species. The set of readings of DNA sequences in the analysis of a gnome is disordered. To obtain the original DNA fragment, all sequences obtained correctly must be fitted. This process is called sequence assembly and aims to determine the correct order of fragments of the

sequenced DNA. The output of the assembly is a set of contigs that are formed by the alignment of the input sequences, see Figure 9 taken from (Cañizares, Blanca and Ziarsolo 2015). The number of contigs is well below the number of fragments of entry, since each contig can be formed by several fragments. When assembling a set of fragments it is unlikely that all the fragments will form part of the resulting contigs. These sequences that do not form any contig are called singletons (Días 2011; Hernandez, Olmos and Olvera 2016).

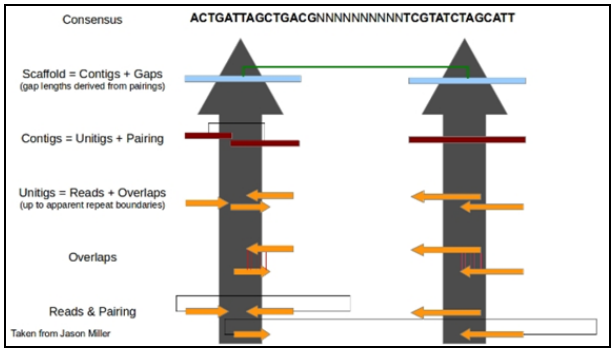

Figure 9: Terminology in the assembly sequence (Cañizares, Blanca and Ziarsolo 2015)

A contig is a set of overlapping DNA segments that together represent a consensus region of DNA. A supercontig is an even longer sequence than a contig, formed by the union of two or more contigs. The union of two or more supercontigs forms a scaffold. Several scaffolds finally create the structure of the chromosome see Figure 10 taken from (Monya 2012).

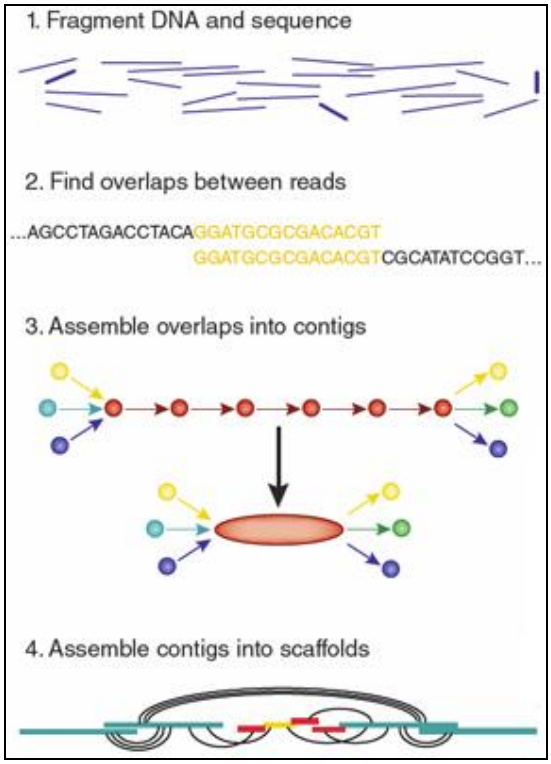

Figure 10: Genome assembly

1. Fragment DNA and sequence: A series of readings are made by the machines called sequencers. Readings are small fragments of DNA.
2. Find overlaps between reads: To assemble the sequence, the splices between the DNA fragments obtained are found. With this information a map is created to assemble longer sequences.
3. Assemble overlaps into contigs: A continuous sequence of DNA is generated that has been assembled through the spliced DNA fragments, and larger fragments of the DNA are being created.
4. Assemble contigs into scaffolds: A sequence formed by the union of one or more contigs is created. In a scaffold it is not required that there is an overlap between contigs (not so for the latter) to assemble longer strings.

6. ASSEMBLY OF DNA SEQUENCES USING CPANS

DNA sequencing is the process of determining the precise order of nucleotides within a DNA molecule. It includes any method or technology that is used to determine the order of the four bases: adenine, guanine, cytosine, and thymine in a strand of DNA (Masoudi-Nejad et al. 2013). The advent of rapid DNA sequencing methods has greatly accelerated biological and medical research and discovery. 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, including the human genome and other complete DNA sequences of many animal, plant, and microbial species (Pareek et al., 2011). In this regard use of CPANs 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 (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 (Blelloch and Guy 1996), (The Myers algorithm performs search matching of substrings using a deterministic automaton: Read all the characters of the text one by one and modify in each step some variables that allow to identify the possible occurrences. Some similar algorithms are: Knuth-Morris-Pratt, Shift-Or, Shift-And).

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, particular splices are located and associated with the processes (a process for each splice sequence of DNA strings). A splice graph is then generated that shows the relationship between pairs of nodes (processes), as well as the lack of communication among others. The set of nodes (processes) 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.

6.1. Parallelization algorithm for the assembly of DNA sequences by an example

Given a genome which we will assume has been sequenced covering most of the structure, but missing a small fragment:

GTAGTCATGTTGAAAAACTTACGAGTAAATTACGTTGTCGA
GGGCGTGCAAGTAGCGCAACCCGTGACAAGCGCAAATTCG
GAAGTAT**ACGCCAA**TCTACCGCCTCCCGTACCCGCGGAGAC
GTATCAAACCGACGAAGATTACGAGGAAGATGACGGAGG
GTGGGC

Figure 11: Sequenced genome

Figure 11 shows the genome where it is assumed that the reading of the part underlining was not obtained. Assembling the readings of this genome is expected to obtain two contigs by not having been able to read the complete sequence (table 1).

Table 1: Read Contigs

Contig 1	Contig 2
GTAGTCATGTTGAAAAACTT	TCTACCGCCTCCCGTACCCGCG
ACGAGTAAATTACGTTGTCG	GAGACGTATCAAACCGACGAA
AGGGCGTGCAAGTAGCGCA	GATTACGAGGAAGATGACGG
ACCCGTGACAAGCGCAAATT	AGGGTGGGC
CGGAAGTAT	

As a result of the readings we obtain the list of fragments shown in table 2, where the splices are highlighted, the prefixes marked in green, the suffixes in red and in bold the splices that are overlapped by a suffix and prefix of other strings. A prefix and a suffix represent strings of length pb. The unmarked parts have no overlap, which indicates that those sections were read only once. The size of the desired splice is selected, for this example a splice of length 6 is used, the prefix and suffix of each string must be compared to the rest of the complete strings.

Table 2: Fragments Read

Process	Fragments read
1	GTAGTCATGTTGAAAAACTTACGAGT
2	ACGAGTAAATTACGTTGTCGAGGGCG
3	AGGGCGTGCAAGTAGCGCAACCCGTG
4	CCCGTGACAAGCGCAAATTCGGAAG
5	CGTGACAAGCGCAAATTCGGAAGTAT
6	TCTACCGCCTCCCGTACCCGCGGAGAC
7	CCGCCTCCCGTACCCGCGGAGACGTA
8	GACGTATCAAACCGACGAAGATTACGA
9	TTACGAGGAAGATGACGGAGGGTGGGC

The first column of the table shows the number of processes needed to process the fragments in a parallel way in the CPAN, which will be grouped later forming the working processes within the FARM of the CPAN, each of which will be constituted by a pipeline of related processes. Each of the nine processes is mapped to work with a string and look for overlaps in the remaining fragments. Two substrings (prefix and suffix) of each fragment are taken to compare them with the other fragments, thus obtaining the results shown in table 3, using the algorithm Myers.

Table 3: Result of the Myers algorithm by fragment

Fragment	Location of the splice found								
	1	2	3	4	5	6	7	8	9
1	-	**5**	27	25	26	27	26	27	27
2	**24**	-	**5**	25	26	27	26	27	27
3	26	**25**	-	**5**	26	27	26	27	27
4	26	25	**26**	-	**23**	27	26	27	27
5	26	25	27	**7**	-	27	26	27	27
6	26	25	27	25	26	-	**22**	27	27
7	26	25	27	25	26	**9**	-	**5**	27
8	26	25	27	25	26	27	**25**	-	**5**
9	26	25	27	25	26	27	26	**26**	-

The calculation of the overlaps is stored in a list for each process, where the incidence of overlaps is added at the end. The results of the exhaustive search are shown in Table 3. The results where splices in the prefix or suffix are found contain the location of the splicing within the indices of the string. Table 4 shows the list that represents the splicing graph constructed using the Myers algorithm. It can be seen that there is no relationship between the pair of nodes (5,6) and there is no arc between the nodes {1,2,3,4,5} and {6,7,8,9}. The graph obtained is shown in Figure 12 and will be represented in the CPAN with a process farm with worker processes and a controller process (figure 13). Each worker process is a CPAN Pipeline that represents each graph of the figure 12. The slave object associated with each CPAN FARM will perform an in-depth search to output the corresponding strings for each group.

ISBN 978-88-97999-85-0; Affenzeller, Bruzzone, Jiménez, Longo and Piera Eds.

Table 4: Graphs with found splices

Process	List of splices	
1	2	-
2	1	3
3	2	4
4	3	5
5	4	-
6	7	-
7	6	8
8	7	9
9	8	-

This is shown in Figure 12. It shows the number of groups formed (graphs) and the elements that belong to each group. For more details see (Hernández, Olmos and Olvera 2016).

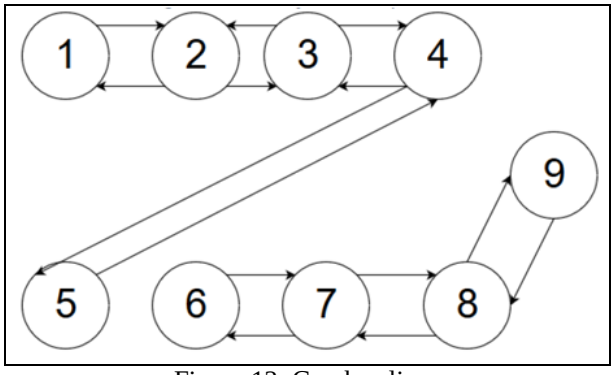

Figure 12: Graph splices

Finally the groups formed by this algorithm are shown in Table 5. This result can be put in a DNA sequence assembler to form the initial contigs of Table 1.

Table 5: Groups obtained by in-depth search

	Group 1		Group 2
1	GTAGTCATGTTGAAAAACTTACGAGT	6	TCTACCGCCTCCCGTACCCGCGGAGAC
2	ACGAGTAAATTACGTTGTCGAGGGCG	7	CCGCCTCCCGTACCCGCGGAGACGTA
3	AGGGCGTGCAAGTAGCGCAACCCGTG	8	GACGTATCAAACCGACGAAGATTACGA
4	CCCGTGACAAGCGCAAATTCGGAAG	9	TTACGAGGAAGATGACGGAGGGTGGGC
5	CGTGACAAGCGCAAATTCGGAAGTAT		

Similarly, the relationships between nodes within each worker FARM process can be represented as the patterns Pipeline. A double communication precisely represents the splices found in the DNA sequences. In Figure 13 is shown the representation of graph-splices as a CPAN.

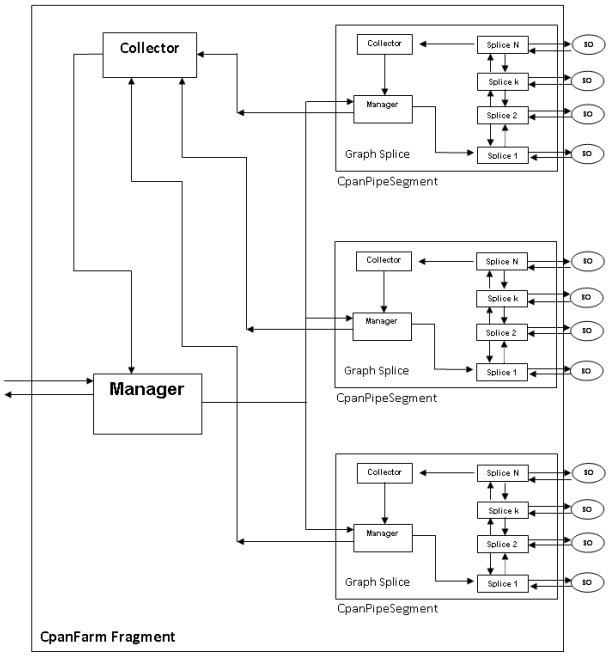

Figure 13: The CpanGraphADN

The new CPAN named CpanGraphADN is structured as a FARM of n-worker processes, i.e., n-fragments of DNA sequences and each worker process is itself a two directions-communication pipeline CPAN formed by m-stages where each stage of CpanPipe 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, a in-depth search is performed to locate these items and obtain the sequence groups formed by the sequential algorithm assigned to each of the CPAN'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. In summary:

1. Two substrings are taken from each fragment of DNA sequences found to compare them with the remaining fragments. The splices are located and are associated with processes within a working process of CPAN Farm, in the new CpanGraphADN of figure 13.
2. The represented graph is processed in a Cpan Farm worker process using a Cpan Pipe and eliminates repetitions in the Cpan Farm Collector process once worker processes send their result to it.
3. It creates the hierarchy contigs, scaffolds, DNA sequence

6.2. CpanGraphADN utility, experiments and results

An experiment was designed by using the CpanGraphADN with genomes of viruses and bacteria available on the web see Table 6, whose data were obtained from European Nucleotide Archive, ENA http://www.ebi.ac.uk. Sequencer readings were simulated to create pseudo-random synthetic readings,

so that the number of contigs can be formed, see (Vera and Gonzalez 2014).

Table 6: Genome used by the CpanGraphADN for experimentation

Genome	Extracted characteristics		
	Length	Number of readings	Genome coverage (average)
Avalone herpesvirus victoria	211519	12691	0.9999
Adoxophyes orana granulovirus	99658	5979	0.9997
Adoxophyes orana nucleopolyehedro	111725	6703	0.9996
African swine fever virus benin	182285	10937	0.9999
Feline coronavirus	29215	1752	0.9998
Shrimp white spot syndrome	307288	18437	0.9999
Trichoplusia ni ascovirus 2c	174060	10443	0.9999
Vaccinia virus GLU-1 h68	203058	12183	0.9998

The CPAN used a fragmented genome, and the resulting fragments were pooled ensuring the same number of groups readings and the follower contigs results were obtained, see table 7. However, it should be noticed that the same number of genomes groups on the contigs was not always obtained. Thus, in the last three contigs that read the genome, only 4 groups were obtained by grouping the simulated fragments readings in CPAN. Similarly, for the second genome only two groups were obtained and only one contig was read. For the remaining genomes, the number of expected readings searches (of different lengths) by the CPAN was obtained.

Table 7: DNA sequence groups found by the CPAN

No.	Genome	Joint length (pb)						Contigs read
		15	17	23	29	40	56	
1	Avalone herpesvirus victoria	2	2	2	4	16	33	2
2	Adoxophyes orana granulovirus	2	2	3	5	5	7	1
3	Adoxophyes orana nucleopolyehedro	1	2	2	2	4	9	2
4	African swine fever virus benin	1	2	3	4	8	23	2
5	Feline coronavirus	1	1	1	1	1	3	1
6	Shrimp white spot syndrome	1	1	1	3	8	35	1
7	Trichoplusia ni ascovirus 2c	1	2	2	3	8	13	2
8	Vaccinia virus GLU-1 h68	1	1	2	4	6	25	3

7. PERFORMANCE

Finally, the execution time of CpanGraphADN is shown in Figure 14 and Scalability of the speedup found is shown in figures 15 to 22. The plot shows the number of processes deployed for the calculation of eight genomes in an experiment conducted on a computer with Intel Core i8 processor, and using a video accelerator card with 1,664 CUDA cores and clock frequency 1,178 MHz.

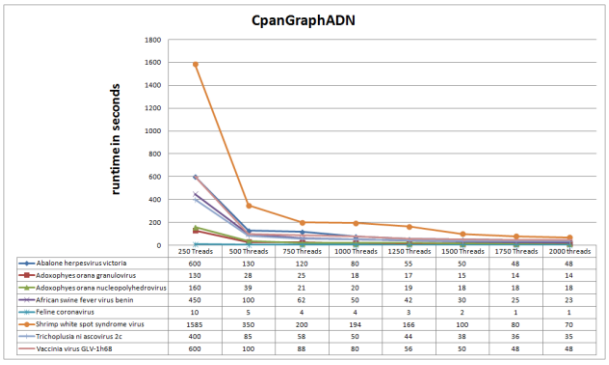

Figure 14: Runtime CpanGraphADN

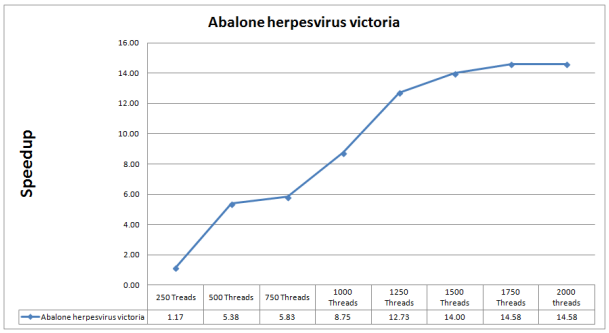

Figure 15: Scalability of the speedup found for the CpanGraphADN for the genome Abalone herpesvirus victoria

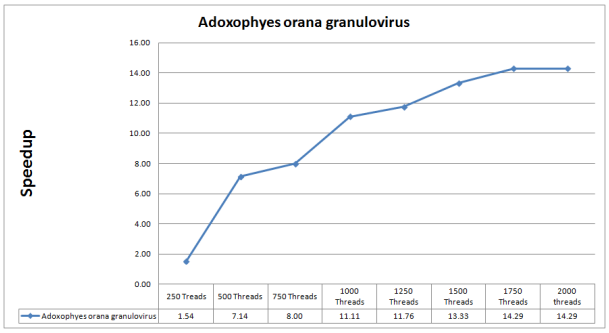

Figure 16: Scalability of the speedup found for the CpanGraphADN for the genome Adoxophyes orana granulovirus

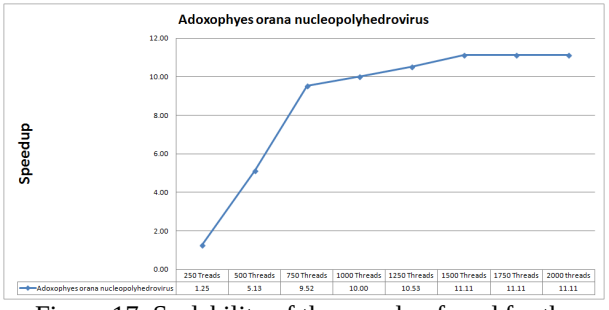

Figure 17: Scalability of the speedup found for the CpanGraphADN for the genome Adoxophyes orana nucleopolyhedrovirus

ISBN 978-88-97999-85-0; Affenzeller, Bruzzone, Jiménez, Longo and Piera Eds.

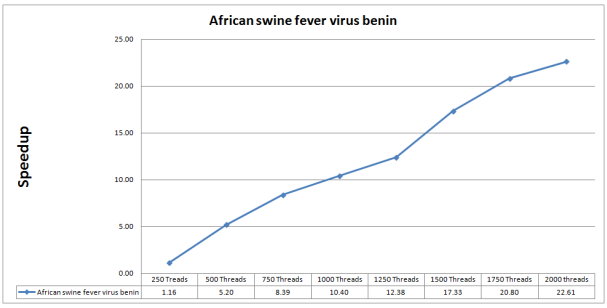

Figure 18: Scalability of the speedup found for the CpanGraphADN for the genome African swine fever virus benin

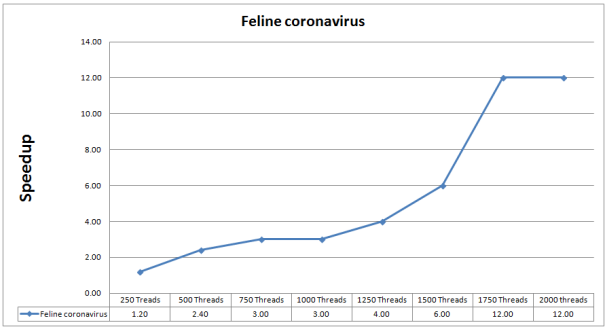

Figure 19: Scalability of the speedup found for the CpanGraphADN for the genome Feline coronavirus

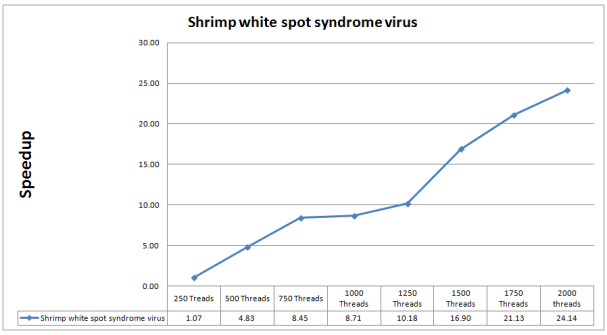

Figure 20: Scalability of the speedup found for the CpanGraphADN for the genome Shirimp white spot syndrome virus

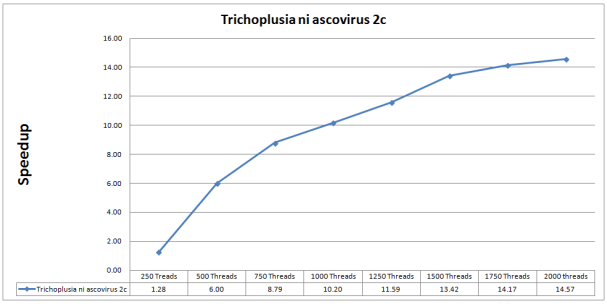

Figure 21: Scalability of the speedup found for the CpanGraphADN for the genome Trichoplusia ni ascovirus 2c

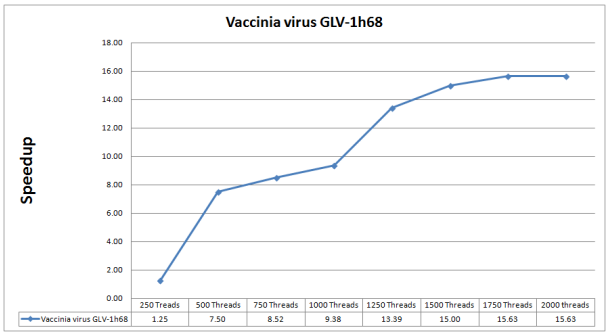

Figure 22: Scalability of the speedup found for the CpanGraphADN for the genome Vaccinia virus GLV-1h68

8. CONCLUSIONS

We have presented a method of design concurrent applications based on the CPAN construction that has been shown susceptible to be used in different platforms (not only with C ++ and POSIX threads). The CPANs have been exercised: Pipe and Farm extracted from a larger library and have shown their utility as communication patterns between concurrent processes, which can be used by programmers with little experience in Parallel Programming. The implemented CPANS can be exploited, thanks to the adoption of the approach oriented to objects. Well-known algorithms that solve sequential problems in algorithms parallelizable have transformed and with them the utility of CPANS has been proven.

We have presented as a case study the parallel calculation of the DNA sequences for 8 genomes and it has been demonstrated efficient, observing Amdahl acceleration, for a restricted range of the number of processors. In the future, more efficient methods will be used to solve the assembly problem: pairing of DNA fragments sequences and voracious algorithms on representation graphs to solve the splices instead of the divide & conquer technique.

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*.

Blelloch, Guy E., 1996. Programming Parallel Algorithms. *Comunications of the ACM*. Volume 39, Number 3.

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

Barry W., Allen M., 1999. Parallel Programming. Techniques and Applications Using Networked Workstations and Parallel Computers. *Prentice Hall*. ISBN 0-13-671710-1.

Cañizares Sales J., Blanca J., Ziarsolo P. 2015. Ensamblaje y mapeo de secuencias tipo Sanger. *Bioinformatics & Genomics Group*. *Polytechnic*

University of Valencia. Valencia, España. https://bioinf.comav.upv.es/courses/intro_bioinf/ensamblaje.html

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.

Danelutto, M.; Orlando, S; et al., 1995. Parallel Programming Models Based on Restricted Computation Structure *Approach. Technical* Report-Dpt. Informatica. Universitá de Pisa.

DanishAli, S. and Farooqui, Z. 2013. Approximate Multiple Pattern String Matching using Bit Parallelism: *A Review. International Journal of Computer Applications*, Volume 74, No.19: pp.47-51.

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

De Simone, et al. 1997. Designs Patterns for Parallel Programming. *PDPTA International Conference*.

Días J.D. 2011. Estrategia de solución al problema de la anotación de secuencias de ADN mediante la metodología CommonKADS. Disertación Master en Inteligencia Artificial. *Facultad de Informática. Universidad Complutense de Madrid*. Madrid, España.

Hernandez R., Olmos I., Olvera A. 2016. Agrupación de fragmentos de secuencias de ADN a partir de Técnicas paralelas para tareas de ensamblaje de genomas. *Disertacion de Tesis de Licenciatura*. Universidad Autónoma de Puebla. México.

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*. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.477.7088&rep=rep1&type=pdf

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

Monya Baker, 2012. De novo genome assembly: what every biologist should know. *Nature America Inc*, Volume 9. No. 4: Pp. 333-337. ISSN: 1548-7091. http://www.nature.com/nmeth/journal/v9/n4/full/nmeth.1935.html

Pareek, C., Smoczynski, R. and Tretyn, A. 2011. Sequencing technologies and genome sequencing. *Journal of Applied Genetics*, Volume 52, No.4: pp.413-435.

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 (CPANs). *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., 2012. Compositions of Parallel Objects to Implement Communication Patterns. *XXIII Jornadas de Paralelismo. SARTECO 2012*. Septiembre de 2012. Elche, España.

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.

Rossainz M, Capel M., Domínguez P., 2015. Pipeline as high level parallel composition for the implementation of a sorting algorithm. *27TH European Modeling & Simulation* Symposium.Campora Bergeggi, Italy.

Samiksha S., 2016. Structure of Chromosome: Size and Share.*YourArticleLibrary.com.* http://www.yourarticlelibrary.com/biology/structure-of-chromosome-size-and-share-474-words/6648/

Vera, F. and González, B. 2014. LPS: una estrategia de ensambles de secuencias cortas de ADN, Disertacion de examen de master. *Instituto Nacional de Astrofsica, Óptica y Electrónica*, Tonantzintla Pue., Mexico.

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