OPTIMIZATION OF A MODULAR NEURAL NETWORK FOR PATTERN RECOGNITION USING PARALLEL GENETIC ALGORITHM

Martha Cárdenas, Patricia Melin, Laura Cruz

Abstract:

In this paper, the implementation of a Parallel Genetic Algorithm (PGA) for the training stage, and the optimization of a monolithic and modular neural network, for pattern recognition are presented. The optimization consists in obtaining the best architecture in layers, and neurons per layer achieving the less training error in a shorter time. The implementation was performed in a multi-core architecture, using parallel programming techniques to exploit its resources. We present the results obtained in terms of performance by comparing results of the training stage for sequential and parallel implementations.

Keywords: *modular neural networks, parallel genetic algorithm, multi-core.*

1. Introduction

The recognition of individuals, from their biometric features, has been driven by the need of security applications, mainly of security such as in surveillance systems for control of employee assistance, access control security places, etc. These systems, have been developed with different biometrics including face recognition, fingerprints, iris, voice, hand geometry, and more [15]. Although there are systems based on classical methods, biometric pattern recognition developed in the area of artificial intelligence with techniques such as fuzzy logic, data mining, neural networks, and genetic algorithms.

Real-world problems are complex to solve and require intelligent systems that combine knowledge, techniques and methodologies from various sources. In this case, we are talking about hybrid systems, and these can be observed in some applications already developed in [6], [13], [14].

Artificial Neural Networks applied to pattern recognition have proved to give good results. Therefore is complex to deal with monolithic neural networks. The use of modular neural networks can divide the complex problem into several task smaller, in order to get a efficient system and good results.

Artificial Neural Networks and Modular Neural Networks have high potential for parallel processing. Their parallel nature makes them ideal for parallel implementation techniques; however it's difficult to find optimal network architecture for a given application. The architecture and network optimal parameter selection is the most important part of the problem and is what takes a long time to find. Genetic Algorithms (GAs) are search techniques that used to solve difficult problems in a wide range of disciplines. Parallel Genetic Algorithms (PGAs) are parallel implementations of GAs, which can provide considerable gains in terms of performance and scalability. PGAs can easily implemented on networks of heterogeneous computers or on parallel machines like a multicore architecture.

Dongarra *et al.* [10] describe several interesting applications of parallel computing. In the coming years, computers are likely to have even more processors inside, and in [3] a description of multi-core processor architecture is presented. An introduction to Multi-Objective Evolutionary Algorithms can also be found in [8].

The primary goal of this research is to implement an optimized modular neural network system for multimodal biometric. In this paper first we describe the implementation of a monolithic neural network optimized with a PGA, and later the first stage of the modular system that consist in a modular neural network for only one biometric measure, the system is optimized using a PGA master-slave synchronous.

The paper is organized as follows: in the section 2 we explain relevant concepts include in this research, section 3 defines the problem statement and the method proposed, section 4 presents the results achieved and finally Section 5 show the conclusions and future work.

2. Theoretical Concepts

Soft Computing consists of several computing paradigms, including fuzzy logic, neural networks and genetic algorithms, which can be combined to create hybrid intelligent systems, these systems leverage the advantages of each of the techniques involved [15]. In this research, we use the paradigms of neural networks and genetic algorithms.

2.1. Artificial Neural Networks

A neural network is a computational structure capable of discriminating and modeling nonlinear characteristics. It consists of a set of units (usually large) of interconnected simple processing, which operate together. Neural networks have been widely used because of their versatility for solving problems of prediction, recognition, approach [6], [15], [12], [24].

These systems emulate, in a certain way, the human brain. They need to learn how to behave (Learning) and someone should be responsible for teaching (Training), based on previous knowledge of the environment problem [16], [26], 25].

The most important property of artificial neural networks is their ability to learn from a training set of patterns, i.e. is able to find a model that fits the data [22].

2.2. Modular Neural Networks

A review of the physiological structures of the nervous system in vertebrate animals reveals the existence of a representation and hierarchical modular processing of the information [18].

Modularity is the ability of a system being studied, seen or understood as the union of several parts interacting and working towards a common goal, each performing a necessary task to achieve the objective [2].

According to the form in which the division of the tasks takes place, the integration method allows to integrate or to combine the results given by each of the constructed modules. Some of the commonly used methods of integration are: Average, Gating Network, Fuzzy Inference Systems, Mechanism of voting using softmax function, the winner takes all, among others.

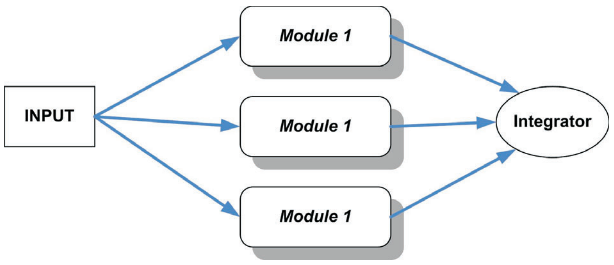

Fig. 1. Modular Neural Network.

In Fig. 1 shows a general diagram of a Modular Neural Network, in this model the modules work independently and in the end a form commonly called integrator, performs the function of deciding between the different modules to determine which of them has the best solution (including network of gateways, fuzzy integrator, etc.) [17].

2.3. Genetic Algorithms

John Holland introduced the Genetic Algorithm (GA) in 1970 inspired by the process observed in the natural evolution of living beings [26], [19]. Genetic Algorithms (GAs) are search methods based on principles of natural selection and genetics. A GA presents a group of possible solutions called a population; the solutions in the population called individuals, each individual is encoded into a string usually binary called chromosome, and symbols forming the string are called genes. The Chromosomes evolve through iterations called generations, in each generation the individuals are evaluated using some measure of fitness. The next generation with new individuals called offspring, is formed from the previous generation using two main operators, crossover and mutation, this representation is shown in Fig 2.

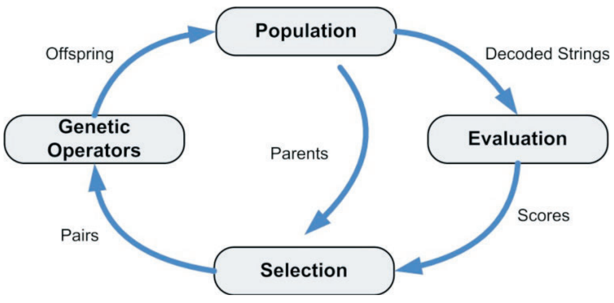

Fig. 2. Structure of a Simple GA.

These optimization techniques are used in several areas such as business, industry, engineering and computer science, also are used as a basis for industrial planning, resource allocation, scheduling, decision-making, etc. The GA is commonly used in the area of intelligent systems, some examples of optimization of fuzzy logic systems and neural networks are shown in [4]. GAs find good solutions in reasonable amounts of time, however, in some cases GAs may require hundreds or more expensive function evaluations, and depending of the cost of each evaluation, the time of execution of the GA may take hours, days or months to find an acceptable solution [4], [19].

Computers with a chip multiprocessor (CMP) give the opportunity to solve high performance applications more efficiently using parallel computing. However, a disadvantage of GAs is that they can be very demanding in terms of computation load and memory.

The Genetic Algorithms have become increasingly popular to solve difficult problems that may require considerable computing power, to solve these problems developers used parallel programming techniques, the basic idea of the parallel programs is to divide a large problem into smaller tasks and solve simultaneously using multiple processors. The effort for efficient algorithms has led us to implement parallel computing; in this way it's possible to achieve the same results in less time. However, making a GA faster is not the only advantage that can be expected when designing a parallel GA. A PGA has an improved power to tackle problems that are more complex since it can use more memory and CPU resources [1].

2.4. Parallel Genetic Algorithms

The way in which GAs can be parallelised depends of several elements, like how the fitness is evaluated and mutation is applied, if single or multiples subpopulations (demes) are used, if multiple populations are used, how individuals are exchanged, how selection is applied (globally or locally).

Existing different methods for implementing parallel GAs and can be classified in the next general classes [20]:

- Master-Slave parallelisation (Distributed fitness evaluation),
- Static subpopulation with migration,
- Static overlapping subpopulations (without migration),
- Massively parallel genetic algorithms,
- Dynamic demes (dynamic overlapping subpopulations),
- Parallel Steady-state genetic algorithms,
- Parallel messy genetic algorithms,
- Hybrid methods.

Our Implementation is based on the Master-Slave Synchronous parallelisation, and for that reason we describe only this method, other methods can be reviewed in [4], [19].

Master-slave GAs have a single population. One master node executes the operator's selection, crossover, and mutation, and the evaluation of fitness is distributed among several workers (slaves) processors. The workers evaluate the fitness of every individual that they receive from the master and return the results. A Master-Salve GA depending on whether it waits to receive the fitness values

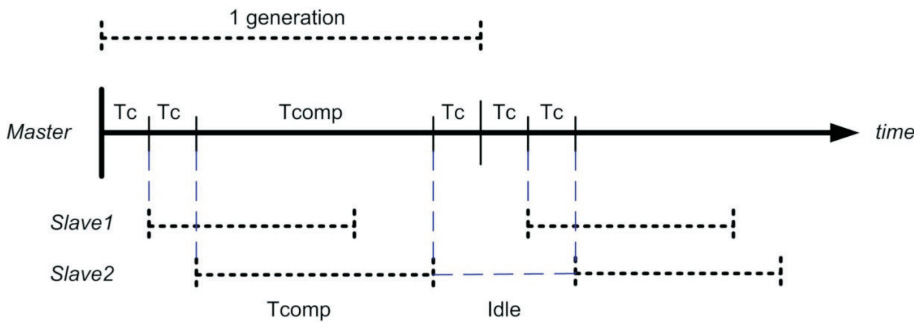

Fig. 3. Execution of Parallel Master-Slaves synchronous GA.

for the entire population before proceeding to the next generation can be synchronous or asynchronous. In Fig. 3 we show a Master-Slaves Synchronous GA.

2.5. Chip Mpltiprocessors (CMPs)

The improvement in actual processors is based on the development of Chip Multiprocessors (CMPs) or Multi-core processors, thus to increase the efficiency of a processor, increases the number of cores inside the processor chip.

Multi-core processors technology is the implementation of two or more "execution cores" within a single processor, some of the advantages of multi-core architectures are shown in [10], [7]. These cores are essentially two or more individual processors on a single chip. Depending on the design, these processors may or may not share a large on-chip cache; the operating system perceives each of its execution cores as a discrete logical processor with all the associated execution resources [5].

CMPs can achieve higher performance than would be possible using only a single core. The low inter-processor communication latency between the cores in a CMP helps make a much wider range of applications viable candidates for parallel execution. The increasing complexity of parallel multicore processors necessitates the use of correspondingly complex parallel algorithms.

However to exploit these architectures is necessary to develop parallel applications that use all the processing units simultaneously. In order to achieve parallel execution in software, hardware must provide a platform that supports the simultaneous execution of multiple threads. Software threads of execution are running in parallel, which means that the active threads are running simultaneously on different hardware resources, or processing elements. Now it is important to understand that the parallelism occurs at the hardware level too.

The improvement measure or speedup takes as reference, the time of execution of a program in a mono-processor system regarding the time of execution of the same program in a multiprocessor or multi-core system, which is represented as follows:

$$\text{speedup} = \frac{t_s}{t_p}, \tag{1}$$

Where t_s is the time it takes to run the program in a mono-processor system and t_p is the time it takes to run the same program in a system with p execution units.

There are many models of parallel programming, the two main choices and the most common are Shared-memory programming and Distributed memory [5], also the parallelism can be implemented in two ways, implicit parallelism, that some compilers perform automatically, these are responsible to generate the parallel code for the parts of the program that are parallel, and the explicit parallelism which is implemented using parallel languages, and the responsible of the parallelism is the programmer, that defines the threads to work, the code of each thread, the communication, etc., this last parallelism gets higher performance.

3. Problem Statement

In this case, we focus on the parallel genetic algorithms for optimizing the architecture of a monolithic neural network and Modular Neural network for recognition of persons based on the face biometry implemented in multi-core processors.

For determining the best architecture and parameters for a neural network there is no particular selection criterion, for example the number of layers and neurons per layer for a particular application is chosen based on experience and to find an optimal architecture for the network becomes a task of trial and error. In addition, there are others methods that with a empirical expression can calculate and determining the architecture of neural network for a specific problem [23].

The database used for this research is The ORL Database of Faces of the Cambridge University Computer Laboratory [9]. This database contains ten different images of 40 persons with gestures, for our implementation not apply any preprocessing for this time, the examples pictures shown in Fig. 4.

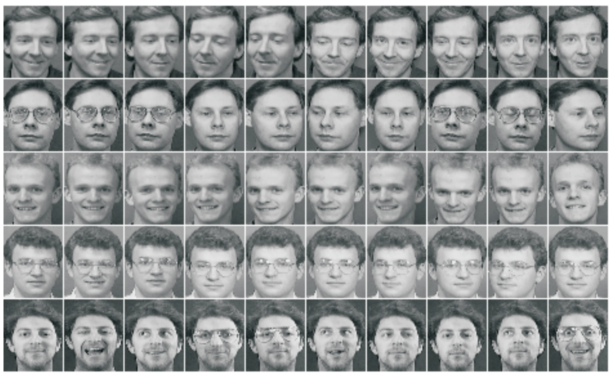

Fig. 4. Some images of the ORL database, the database is composed by 400 images, there are images of 40 different persons (10 images per person).

For the implementation monolithic and Modular, the database was the same, and the same structure of chromosome for optimization.

3.1. Monolithic Neural Network Implementation

Neural networks were applied to a database of 40 persons, and we used 5 images per person for training and 5 images per person for test. First, we implemented the traditional monolithic neural network, and before we implemented a Parallel GA for optimizing layer and neurons per layer. The training method for the neural network is the Trainscg (Scaled Conjugate Gradient), with an error goal of 0.01e-006 and between 100 and 150 generations.

The Genetic Algorithm was tested in a Multi-core computer with following characteristics: CPU Intel Core 2 Quad 2.4 GHz, Bus 1066 MHz, 8MB of L2 cache, Memory 6 GBytes DDR2 of main memory, all the experiments were achieved in the MatLab Version R2009b using the Parallel computing toolbox.

Parallel Genetic Algorithm for Optimization

The Master-Slave Parallel genetic Algorithm was codified with a binary chromosome of 23 bits, 2 bits for number of layers, and 7 bits for number of neurons per layer. The maximum number of layers is 3 and neurons 128, this is shown in Figure 5. The proposed algorithm was implemented in a Shared Memory Multi-core machine with 4 cores, taking one core as master and the remaining cores as slaves.

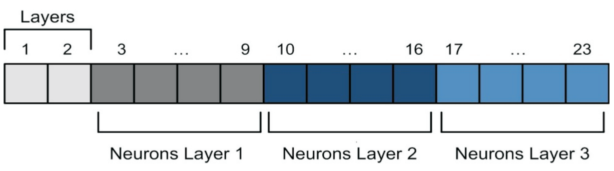

Fig. 5. Chromosome representation of the problem.

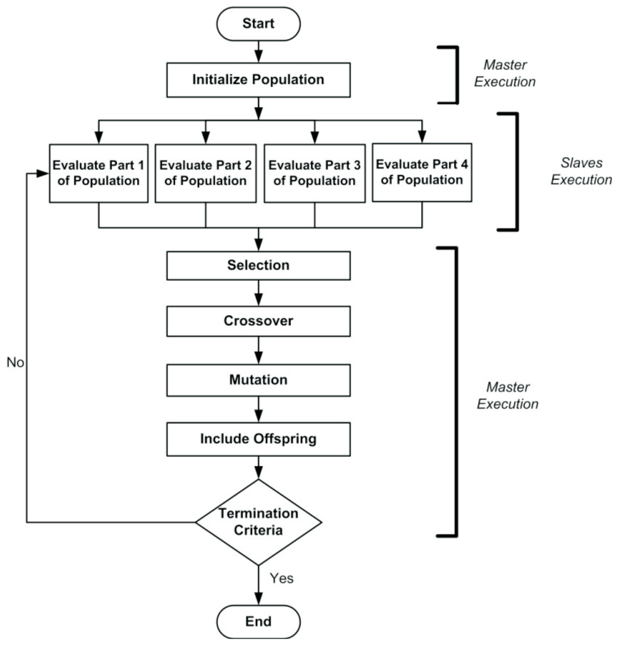

Fig. 6. Parallel GA Implementation.

The Genetic Algorithm has the following characteristics:

- Chromosome Size: The number of genes in each individual for this application is 23 binary bits.
- Population size: Defines the number of individuals that will compose the population.
  Population Size =20
- Termination Criteria: Maximum number of generations for solving the problem.
  Max Generations=50
- Selection: We used Stochastic Universal Sampling
  Selection Prob=0.9
- Crossover: The selected individuals have a probability of mating, acting as parents to generate two new individuals that will represent them in the next generation. The crossing point is random with a probability of 0.7.
- Mutation: Represents the probability that an arbitrary bit in the individual sequence will be changed from its original stat. Mutation Probability 0.8

The flow chart of Fig. 6 shows the parallel GA implementation. Other methods for solving a Parallel GA can be seen in [21], [11], [4].

3.2. Modular Neural Network Implementation

For the modular neural network implementation, we develop a previous stage of normalization for the database; this stage is a parallel one like the training stage. To the original database we apply an algorithm for standardize the size of the image. Depending on the database, if necessary, applies this stage.

In Fig. 7, show the general diagram of the system, the original database it's distributed in all the slaves available for normalization to form a new database, that is the input to the modular neural network. In the system, we have a synchronization step that execute the master to help coordinate the process in all the slaves.

When we have the inputs of the system, the PGA start creating a random initial population in the stage of synchronization, the master divides the population and send it to the slaves (in this case the cores of processor), and the slaves take a part of the population to evaluate.

In all the Slaves, for each individual of the GA, load the architecture of the network, read the images and put as input in the Module that corresponds. The images are propagate in the network and calculate the error of training. When finish the part of population the slaves send the results of evaluation to the master. Master waits to all slaves finish to collect the entire population and make selection based on the fitness of all the individuals.

The Master performs the operators of crossover, mutation and generates the new population to be divided and evaluated in the slaves, until the maximum number of generations is reached. We used the method of integration Gating Network.

The modular architecture consists in dividing the database between the modules or core processors available. The experimental results achieved with this PGA implementation presented in the following section.

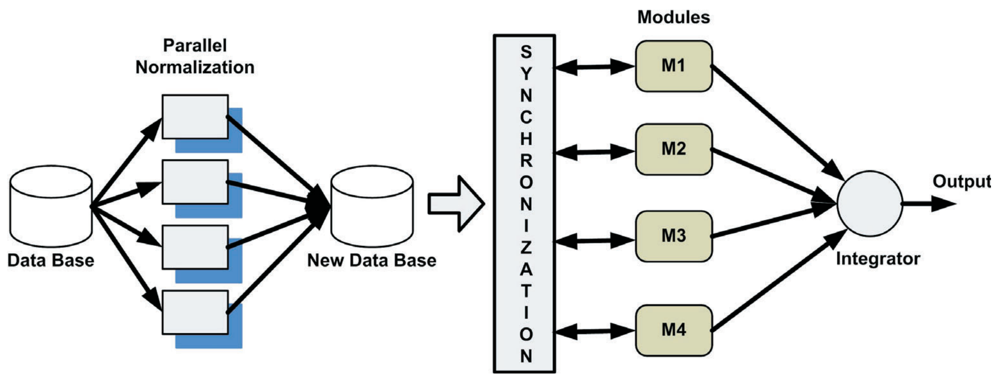

Fig. 7. General Diagram of the Modular Neural Network System.

4. Results

Different experiments developed to observe the performance of parallel genetic algorithms; the results presented in this section, the results in time represent the average time execution of each neural network in the population of the PGA.

4.1. Monolithic Neural Network

First, we train the monolithic neural network without optimization, in sequential form. After manually changing the architecture for several times, we defined the architecture of the neural network with the expression of Salinas [23] as follows:

- First hidden layer $(2 * (k+2)) = 84$.
- Second hidden layer $(k+m) = 45$.
- Output layer $(k) = 40$.

where k corresponds to the number of individuals and m to the number of images of each of them.

Table 1 shows the average of 20 trainings in sequential form of the network in a dual-core and quad-core machines, in this experiment we enabled only one of the cores available and one thread of execution in the processor for simulating sequential execution. Fig. 8 shows the usage of a dual-core machine in the training of the network.

Table 1. Average of 20 trainings in sequential form of the network for dual-core and quad-core machines.

No. Cores	Epochs	Error Goal	Error	Total Time
4	150	1.00E-06	0.00178	1:18 min
2	150	1.00E-06	0.00468	1:14 min

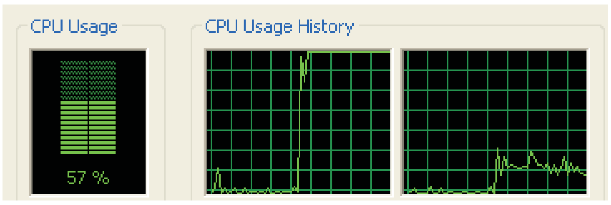

Fig. 8. Cores Usage in the sequential training of the network.

In the experiment of training with implicit parallelism without optimization all the cores and threads available are enabled. Matlab R2009b uses as a default the implicit parallelism for the applications run on it and take all the cores for execution automatically generating a thread of execution per processor. The results obtained for Matlab in a dual-core and quad-core machines shown in the Table 2.

Table 2. Average of 20 trainings of implicit parallelism form of the network for dual-core and quad-core machines.

No. Cores	Epochs	Error Goal	Error	Total Time
4	150	1.00E-06	0.00113	1:58 min
2	150	1.00E-06	0.00384	1:41 min

The results show that the execution of serial training of the network are more efficient that the implicit paralellism of Matlab, because when a single core is working (Fig. 9) all the cache memory is available for them.

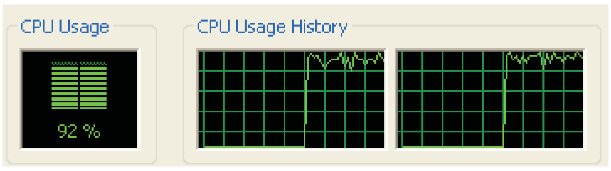

Fig. 9. Cores Usage in the implicit parallelism in training of the network.

Implicit Parallelism with GA optimization

We optimize the monolithic Neural Network with a simple GA in the form of implicit parallelism. Table 3 shows the average of 20 training test for 2 and 4 cores. Figures 10 and 11 show the usage of processor in dual-core and quad-core machines.

Table 3. Average of 20 training of implicit parallelism of Simple GA for optimization of the network with for dual-core and quad-core machines.

N. Cores	Ind	Gen	Cross	Mut	Error	Time/ nework	Average time
2	20	30	0.7	0.8	3.0121e-004	1:51min	33 min
4	20	30	0.7	0.8	9.7361e-005	4:10 min	83 min

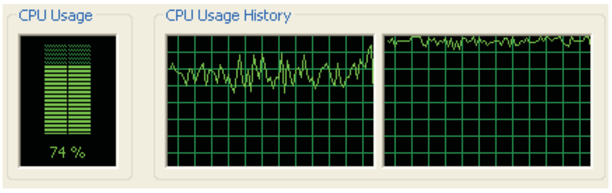

Fig. 10. CPU Performance Implicit Parallelism with 2 cores.

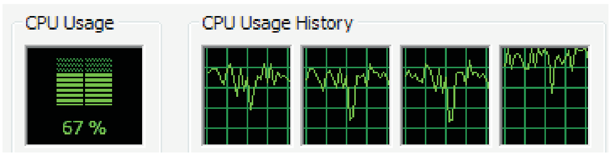

Fig. 11. CPU Performance Implicit Parallelism with 4 cores.

Explicit Parallelism with Parallel GA optimization

In this experiment, we utilize the matlab pool that enables the parallel language features within the MATLAB language by starting a parallel job, which connects this MATLAB client with a number of labs. The average results for 20 executions are shown in table 4. Fig. 12 and 13 show the usage of the processor for training with explicit parallelism in a dual-core and quad-core machines.

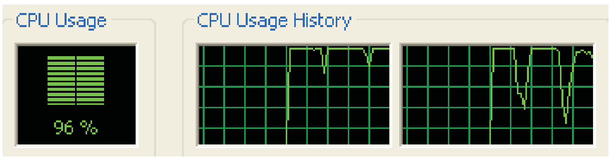

Fig. 12. CPU Performance Explicit Parallelism with 2 cores.

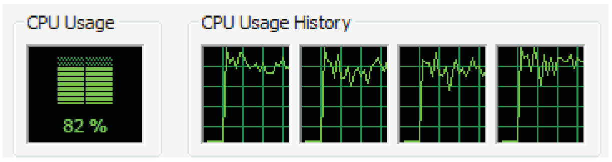

Fig. 13. CPU Performance Explicit Parallelism with 4 cores.

Table 5 shows a comparison between all the training experiments, and observed that.

4.2. Modular Neural Network

The results presented in this section are in Table 6 that shows the average of 10 trainings in sequential form of the Modular network in a dual-core and quad-core machines.

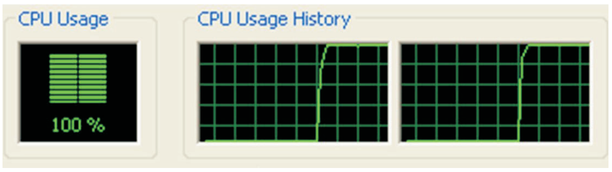

Fig. 14. CPU Performance Explicit Parallelism with 2 cores modular neural network.

In the parallel implementation of the modular neural network utilize the Matlab pool starting a parallel job. The average results for 10 executions are shown in the Table 7. Fig. 15 shows the performance of CPU for the parallel training of a Modular Neural Network.

Table 4. Average of 20 training of explicit parallelism of PGA for optimization of the network with for dual-core and quad-core machines.

N. Cores	Ind	Gen	Cross	Mut	Error	Time/ nework	Average time
2	20	30	0.7	0.8	3.3121e-004	1:03min	21 min
4	20	30	0.7	0.8	4.5354e-004	0:35 sec	12 min

Table 5. Table of Results of experiments Sequential and Parallel per network.

	RNA		**GA-RNA**			
	Sequential		GA. Implicit Parallelism		GA. Explicit Parallelism	
No. Cores	2	4	2	4	2	4
Average time/ network	1:18 min	1:14 min	1:58 min	1:41 min	1:03 min	0:35 sec

Table 6. Average of 10 trainings in sequential form of the modular network for dual-core and quad-core machines.

N. Cores	Epochs	Error Goal	Error	Time p/red	Time p/gen
2	150	1.00E-06	0.0005	2:13 min	20:11 min
4	150	1.00E-06	0.0031	2:05min	20:49 min

Table 7. Average of 10 training of explicit parallelism of PGA for optimization of the modular network with for dual-core and quad-core machines.

N. Cores	Epochs	Error Goal	Error	Time p/red	Time p/gen
2	150	1.00E-06	0.00012	0:58 min	9:37 min
4	150	1.00E-06	0.00025	1:40 min	16:36 min

Table 8. Results of experiments Sequential and Parallel per network and the speedup.

	Modular Network Sequential	**PGA-Modular Network** GA. Explicit Parallelism	
No. Cores	Time	Time	Speedup
2	2:05 min	1:40 min	1.46
4	2:13 min	0:58 sec	3.67

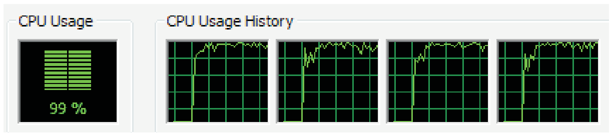

Fig. 15. CPU Performance Explicit Parallelism with 4 cores modular neural network.

In Table 8 we show the results of a modular neural network with a sequential training and with a parallel training, for this experiment modules of the modular network are trained in parallel, the optimization of the network is done in about half (dual core) or about a quarter (quad-core) time, we obtain a speedup of 1.46 in a dual core machine and 3.67 in a Quad core machine. This are the first results obtained, we continue with experiments.

5. Conclusions and Future Work

We have presented the experiments with training of the monolithic and modular neural network for database of face; we used different implementations of parallelism to show that the parallel GA using multi-core processor offers best results in the search for optimal neural network architectures in less time.

The genetic algorithms take considerable time to successfully complete convergence depending of application, but most of the times achieve satisfactory optimal solutions. Genetic Algorithms can be parallelized to speed-up its execution; and if we use Explicit Parallelization we can achieve much better speedup than when using implicit Parallelization, anyway it's necessary to make more tests.

Multi-core computers can help us solve high performance applications in a more efficient way by using parallel computation. The future work consists in considering larger size databases and implementing a modular neural network in a multi-core cluster applying different techniques of parallel processing.

AUTHORS
Martha Cárdenas*, Patricia Melin, Laura Cruz - Tijuana Institute of Technology, Tijuana, México. E-mails: mc.marthacardenas@gmail.com, pmelin@tectijuana.mx.
* Corresponding author

References

[1] E. Alba, A. Nebro, J. Troya, "Heterogeneous Computing and Parallel Genetic Algorithms", *Journal of Parallel and Distributed Computing*, vol. 62, 2002, pp. 1362-1385.
[2] C. Baldwin, K. Clark, *Design Rules, Vol. 1: The Power of Modularity*, Mit Press, Cambridge, 2000.
[3] T.W. Burger, Intel *Multi-Core Processors: Quick Reference Guide*, http://cachewww.intel.com/cd/00/00/20/57/205707_205707.pdf
[4] E. Cantu-Paz, *Efficient and Accurate Parallel Genetic Algorithms*, Kluwer Academic Publisher, 2001.
[5] M. Cárdenas, J. Tapia, O. Montiel, R. Sepúlveda, "Neurofuzzy system implementation in Multicore Processors", *IV Regional Academic Encounter*, CITEDI-IPN, 2008.
[6] O. Castillo, P. Melin, "Hybrid intelligent systems for time series prediction using neural networks, fuzzy logic and fractal theory", *IEEE Transactions on Neural Networks*, vol. 13, no. 6, 2002.
[7] L. Chai, Q. Gao, D.K. Panda, "Understanding the Impact of Multi-Core Architecture in Cluster Computing: A Case Study with Intel Dual-Core System". In: *The 7th IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2007).* May 2007, pp. 471-478.
[8] C.A. Coello, G.B. Lamont, D.A. Van Veldhuizen, *Evolutionary Algorithms for Solvin Multi-Objective Problem*, Springer: Heidelberg, 2004.
[9] The Database of Faces, Cambridge University Computer Laboratory, http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html
[10] J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon, A. White, *Sourcebook of Parallel Computing*, Morgan Kaufmann PublishersL San Francisco, 2003.
[11] S. González, *Optimization of Artificial Neural Network Architectures for time series prediction using Parallel Genetic Algorithms*. Master thesis, 2007.
[12] K. Hornik, "Some new results on neural network approximation," *Neural Networks*, vol. 6, 1993, pp. 1069-1072.
[13] A. Jeffrey, V. Oklobdzija, *The computer Engineering Handbook, Digital Systems and Aplications*, 2nd edition, CRC press, 1993.
[14] P. Kouchakpour, A. Zaknich, T. Bräunl, *Population Variation in Genetic Programming*, Elsevier Science Inc, ISSN:0020-0255, 2007.
[15] P. Melin, O. Castillo, *Hybrid Intelligent Systems for Pattern Recognition using Soft Computing: An Evolutionary Approach for Neural Networks and Fuzzy Systems*, Springer, 2005.
[16] P. Melin, O. Castillo, *Hybrid Intelligent Systems for Pattern Recognition Using Soft Computing*. Springer, Heidelberg, 2005.
[17] B. Morcego, *Study of modular neural networks for modeling nonlinear dynamic systems*, PhD thesis, Universitat Politecnica de Catalunya, Barcelona, Spain, 2000.
[18] I. Quiliano, *Sistemas Modulares, Mezcla de Expertos y Sistemas Híbridos*, February 2007, in Spanish, http://lisisu02.usal.es/~airene/capit7.pdf
[19] M. Mitchell, *An introduction to genetic algorithms*, MIT Press, 1998.
[20] M. Nowostawski, R. Poli, "Parallel Genetic Taxonomy", In: Proceedings of the Third International Conference in Knowledge-Based Intelligent Information Engineering Systems, December 1999, pp. 88-92. DOI: 10.1109/KES.1999.820127.
[21] A. Ross, K. Nandakumar, A.K. Jainet, *Handbook of Multibiometrics*, Springer 2006.
[22] P. Salazar, "Biometric recognition using techniques of hand geometry and voice with computer vision for feature extraction, neural networks and fuzzy logic ", Master thesis, *Division of Graduate Studies and Research in Computer Science*, ITT, 2008, p. 57.
[23] R. Salinas, *Neural Network Architecture Parametric Face Recognition*, University of Santiago of Chile, 2000, pp. 5-9. http://cabierta.uchile.cl/revista/17/

artículos/paper4/index.html

[24] R. Serrano, *Multicore computing applied to Genetic Algorithms*. Master. Thesis, CITEDI-IPN, 2008.

[25] M. Shorlemmer, "Basic Tutorial of Neural Networks". In: *Artificial Intelligence Research Institute*, Barcelona, Spain, 1995.

[26] M. Soto Castro, *Face and Voice Recognition in real time using Artificial Neural Networks*, Master Thesis, Tijuana Institute of Technology, 2006.