Parallel Face Recognition Processing using Neocognitron Neural Network and GPU with CUDA High Performance Architecture

Gustavo Poli and José Hiroki Saito
Libertas Integrated Schools
Federal University of São Carlos

1. Introduction

This chapter presents an implementation of the Neocognitron Neural Network, using a high performance computing architecture based on GPU (Graphics Processing Unit). Neocognitron is an artificial neural network, proposed by Fukushima and collaborators, constituted of several hierarchical stages of neuron layers, organized in two-dimensional matrices called cellular plains. For the high performance computation of Face Recognition application using Neocognitron it was used CUDA (Compute Unified Device Architecture) as API (Application Programming Interface) between the CPU and the GPU, from GeForce 8800 GTX of NVIDIA Company, with 128 ALU's. As face image databases it was used a face database created at UFSCar (Federal University of São Carlos), and the CMU-PIE (Carnegie Melon University - Pose, Illumination, and Expression) database. The load balancing through the parallel processing architecture was obtained by means of the distributed processing of the cellular connections as threads organized in blocks, following the CUDA philosophy of development. The results showed the viability of this type of device as a massively parallel data processing tool, and that smaller the granularity of the parallel processing, and the independence of the processing, better is its performance.

2. Motivation

The face recognition using machines is an active and actual research area. This is composed by multiple disciplines as image processing, pattern recognition, computer vision, artificial neural networks, and computer architectures. There are many commercial applications that implement Face Recognition Techniques, as in access control, and security using video camera.
Many countries use the face recognition techniques for several purposes. On China, for example, it was developed the immigrant recognition system to the cities of Shenzhen and Zhunhai (Terra, 2006). The system gives good results, especially when the fingerprint can´t be used to the recognition purpose, due to several problems as age, damage with chemical reactions, and so on.

Owing to the user-friendly (non-intrusive) property, the face recognition is attractive, despite of the extremely reliable methods of personal biometric identification such as fingerprint and iris scanning analysis.
As it can be seen there are major challenges on the issues of facial recognition, where you can highlight a relationship between two basics variables of the process: the degree of reliability/robustness of the technique being used and computational cost of this technique.
The goal of this chapter is the presentation of a computer architecture for face recognition, aiming its performance increasing through the use of a massively parallel data processing, achieved by the implementation of a Neocognitron neural network architecture, based on GPU (Graphic Processing Unit). To access the GPU as a device for scheduling purposes, it is used in majority the CUDA (Compute Unified Device Architecture), a library that extends the functions of language C, FORTRAN and Python in order to provide the GPU as a device for data processing.

3. Neocognitron Neural Network

Neocognitron is a massively parallel neural network, composed by serveral layers of neuron cells, proposed by Fukushima (Fukushima end Miyake, 1982)(Fukushima and Wake 1992)(Saito and Fukushima, 1998). In a brainway computer it corresponds to part of the human visual recognition system.
The Neocognitron neural network has the basic principle of operation extracting features in a hierarchical manner, i.e., performs the extraction of features in various stages. In the first stage, the extracted features are the simplest; and at the following stages, summing up the lines in different senses of rotation, the features will be presenting with more complexity. The characteristic of this network is that the features extracted by a stage have the informations only sent by the previous stage, as a feedforward neural network.

3.1 The Neocognitron Structure

The stages of a Neocognitron network are arranged in tiers, each of these layers has its own type/complexity of data being processed, and these consist of simple cells (Cell-S), complex cells (Cell-C) and activity cells (Cell-V).
The stages are compared by the Layer-S, of Cell-Ss. The Layer-C, of Cell-Cs; and Layer-V, of Cell-Vs. Within each layer there is a number of cell-plans, which are organized as two-dimensional array of cells, each cell with the ability of extracting the same features of the adjacent cells in the same cell-plan.
The stages function as a tool for organizing the process of extracting the characteristics or factors with a degree of complexity of the extracted pattern characteristics. The first stage, called the zero stage (Stage 0) is not used within the hierarchical scheme of feature extraction and it is used as the retina of the eye, capturing the pattern to be processed by the network. Figure 1 shows the stages of a Neocognitron with five stages.
The number of stages of a Neocognitron network depends on the size of the input pattern being processed by the network. The larger the size of the input pattern, greater is the number of stages required by the network. For example, an input pattern of 20 x 20 pixels, typically results in a network of three hierarchical stages.

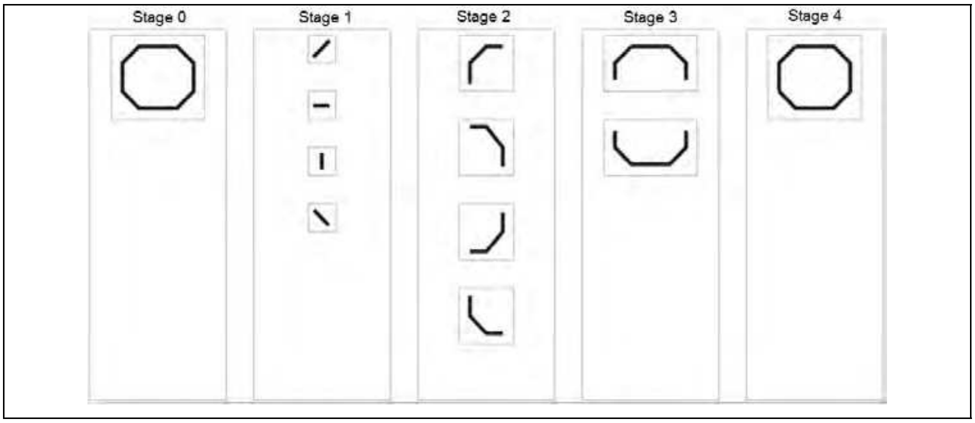

Fig. 1. Neocognitron representation with five stages.

Each stage of a Neocognitron network is divided into three layers: a simple layer (Layer-S), a complex layer (Layer-C) and a layer of activity (Layer-V). Assuming the Neocognitron with five stages, shown earlier (Figure 1), its representation in layers can be seen in Figure 2.

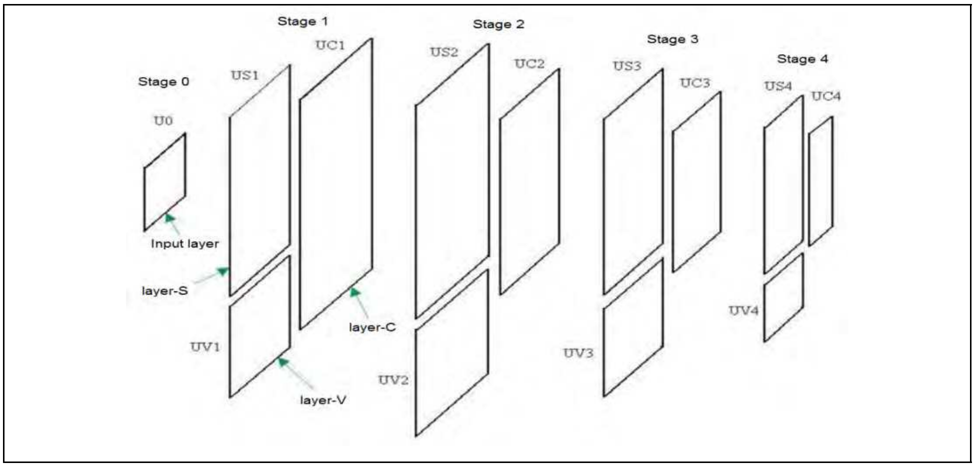

Fig. 2. Neocognitron representation with five stages with its layers.

In stage 0 there is only one layer, which is the input layer or input pattern. All other stages have three types of layers, one Layer-S, a Layer-V and a Layer-C.
Each layer is formed by a number of cellplans. The number of plans in each Layer-S and Layer-C is related to the number of features extracted by the stage of the network. A Layer-V is a single cell-plan layer. The size of the plans is equal to the same layer and it decreases as you climb the hierarchy of stages. Figure 3 shows the plans distributed in the cell layers of the network.

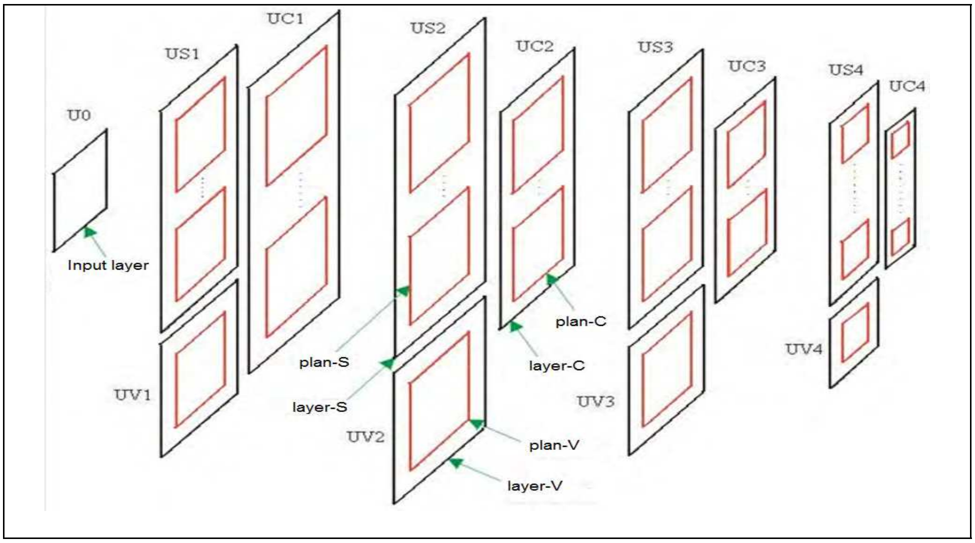

Fig. 3. Five stages Neocognitron representation with its layers and plans.

Each Plan-S, Plan-V, Plan-C and Input layer is formed by a set (array) of specialized cells. Figure 4 shows the cells distributed along the plans of the network. A Plan-C of the layer U_{C4}, the last stage of the network, contains only a single cell, whose activity indicates the recognition of the input pattern.

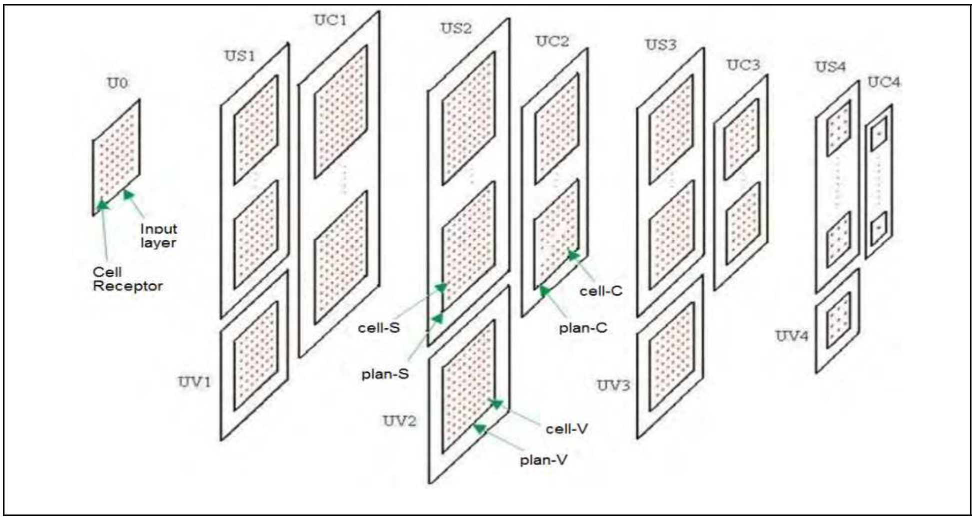

Fig. 4. Five stages Neocognitron representation with its layers, plans, and cells.

3.2 Weights and Connections

A characteristic of the Neocognitron is to have a large number of cells but a reduced number of connections. The cells are connected to a reduced connection area, of the previous layer. This characteristic of connectivity is different from the Multilayer Perceptron, in which a neuron of a layer is connected to all neurons of the previous layer.
For each connection there is a weight that is used to influence the amount of information that is transferred. Neocognitron has four types of weights: weight-a, weight-b, weight-c, and weight-d, whose uses are summarized as shown in Figure 5.

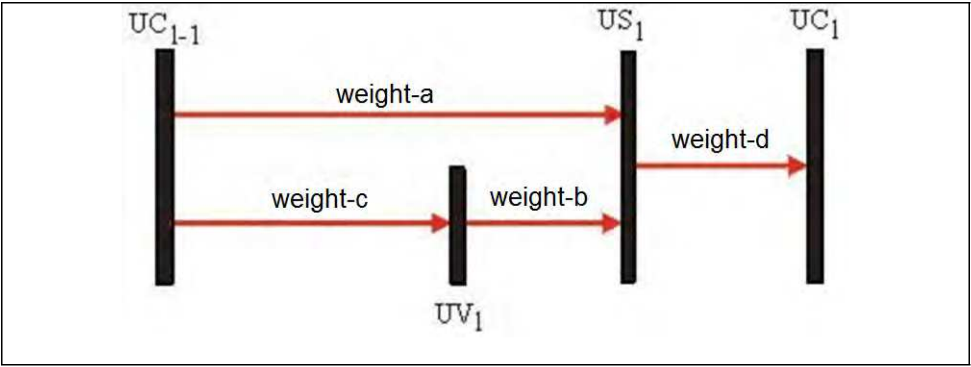

Fig. 5. Neocognitron weights (weight-a, weight-b, weight-c and weight-d) and its connections.

Within a cell-plan level, all cells share the same weight. This causes all cells in the same plan to observe the same feature, thus specializing the plan for the same feature in different positions.

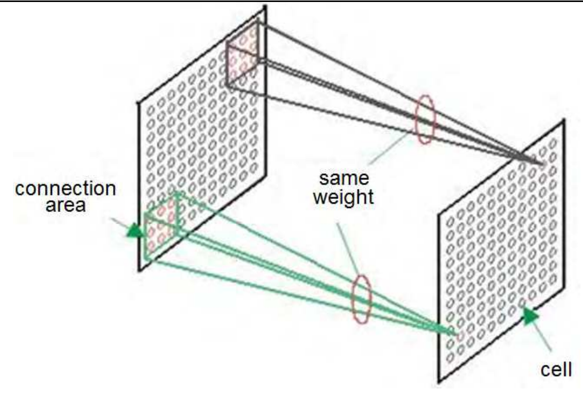

Fig. 6. Within a cellular level, all cells share the same weights.

You can even arrange the weights in two categories, which are modified by training (weight-a weight-b); and which are not modified, i.e., the values attributed to them, remain unchanged through the implementation of the network (weight-c and weight-d).

3.3 Processing Neocognitron Network

Each Cell-V calculates the input values of Cell-Cs from a small region connection area of all cell-plans of the previous Layer-C. The size of the connection area is the same for cells-V and cells-S in a stage of the network and it is determined at the time of construction of the network. An example of the connection area can be seen in Figure 7.

The value of a Cell-V represents the average activity of cells belonging to its area of connection and is used to inhibit the corresponding Cell-S. The exact specification of the function of Cell-V, $u_{Vl}(n)$, is given by Equation 1:

$$u_{vl}(n) = \sqrt{\sum_{k_{l-1}=1}^{K_l} \sum_{i \in S_l} C_l(i), u^2_{C_{l-1}}(n+i, k_{l-1})} \qquad (1)$$

where the weight should be $c_l \geq 0$; and u_{cl}-1(n+i, k_{l-1}) represents the input value, from the previous cell-plan k_{l-1} at the position n+i. Here, i represents a position in a region S_l in a cell-plan.

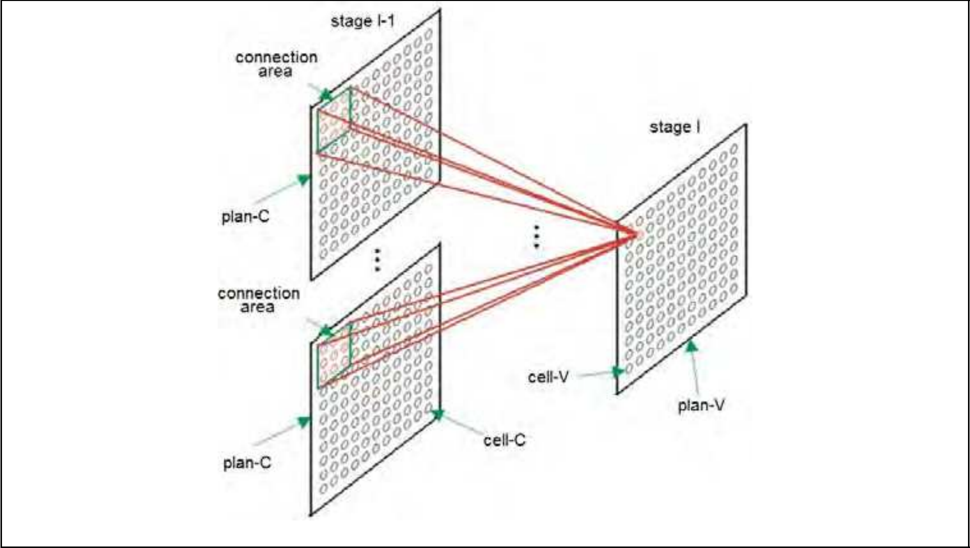

Fig. 7. Example of the connection area of a Cell-V.

The Cell-S evaluates the output values of cell-Cs in a connection area of all cell-plan of the Layer-C of the previous stage, or the input layer. As seen in the previous section, the size of the connection area is the same for cell-Ss and cell-Vs on the same stage (Figure 8).

The role of Cell-S is to recognize a feature in the connected area. To recognize a feature, a Cell-S uses the information in the connection area and information about activities in this area, informed by Cell-V. The feature extracted by a Cell-S is determined by weights on their input connections.

The feature extraction by a plan-S and the significance of the weights is easier to be observed in the cell layer U_o (first layer) of the network. In each cell of layer-S, U_{S1}, following the first layer, there is only one connection area and this area is the receptive field or area of connection of input pattern. Because all cells are equal, any cell in the same cell-plan can recognize the same feature. In the example, the feature is a vertical line that can be in different positions. So, the Cell-S, that is positioned in the connection area containing the feature (vertical line), responds, as outlined in the Plan-S in Figure 9.

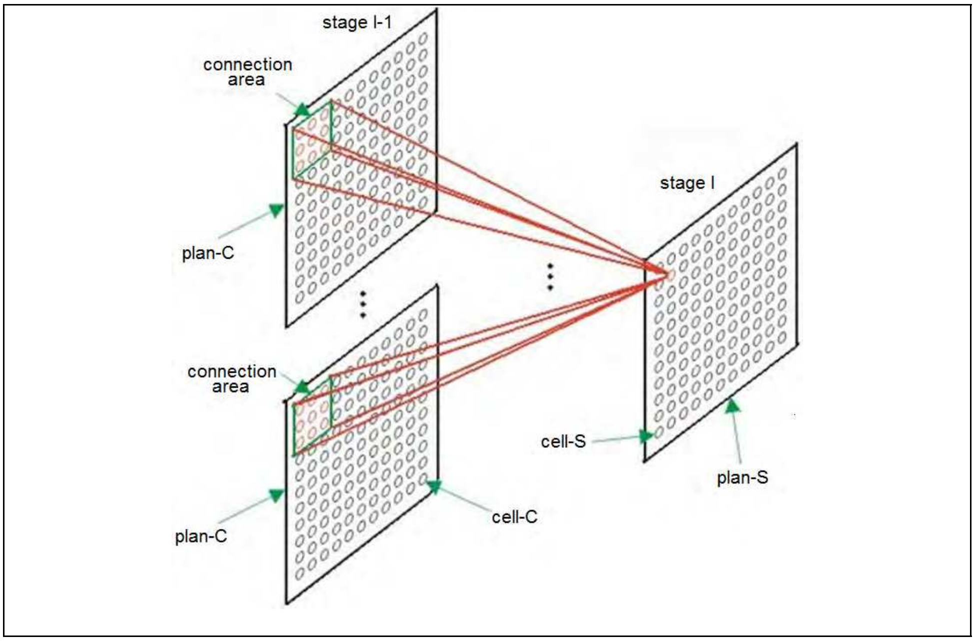

Fig. 8. Example of the connection area of a Cell-S.

The output value of a Cell-S is determined by Equation 2:

$$u_{Sl}(n,k_l) = \frac{\theta}{1-\theta} \cdot \varphi \left[\frac{1 + \sum_{k_{l-1}=1}^{K_{l-1}} \sum_{i \in S_l} a_l(k_{l-1},i,k_l), u_{cl-1}(n+i,k_{l-1})}{1+\theta, b_l(k_l), u_{vl}(n)} - 1 \right] \tag{2}$$

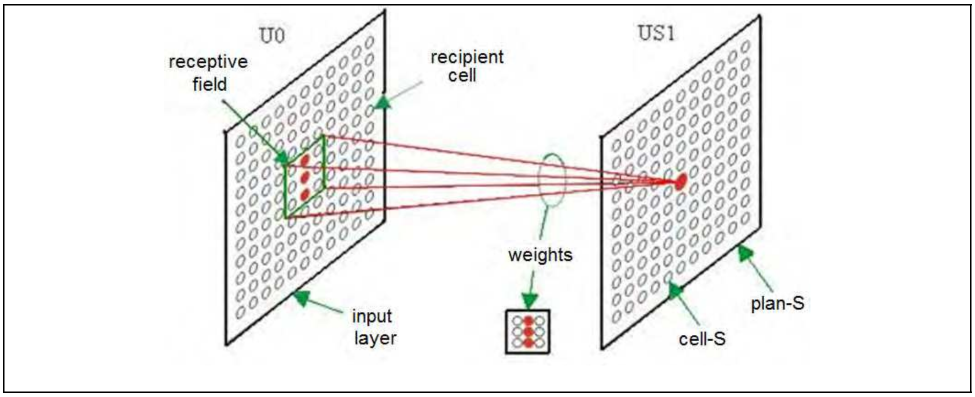

Fig. 9. Example of the connection area of a Cell-S.

The element θ is the threshold parameter with which you can modify the ability for Cell-S extract a particular feature. The weight-a, $a_l(k_{l-1},i,k_l)$, should be greater than or equal to zero, as well as weight-b, $b_l(k_l)$, and the activation function $\varphi[x] = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$.

The Cell-Ss have the ability to extract features not only trained but also distorted, or generalized. This capacity is influenced by the choice of parameter θ, called threshold. It is easy to understand, because the threshold θ multiplies the weighted value coming from the Cell-V, the denominator of the argument. Thus, the lower the value of θ, greater the ability of generalization of trained features.

The cell-C evaluates the output values of plan-S of earlier layer-S (Figure 10). The value of Cell-C depends on the activity of Cell-Ss in its area of connection. The greater the number of active Cell-Ss, greater is the activity of Cell-C. The equation of the Cell-C is described by Equation 3.

$$u_{Cl}(n,k_l) = \psi\left[\sum_{i \in S_l} d_l(i), u_{Sl}(n+i,k_l)\right] \tag{3}$$

As the weight-d, $d_l(i) \geq 0$ and $\psi[x] = \begin{cases} \frac{x}{1+x} & x \geq 0 \\ 0 & x < 0 \end{cases}$.

If a cell-C is active for a single cell-S, all the adjacent cells will be active, so that plan-C contains a blurred representation of the Plan-S. Moreover, as the blurring results in the cell-plan's adjacent values are very close, a small number of Cell-Cs is necessary for the next stage. This results in reducing the size of the Plan-C, in relation to the Plan-S, Figure 11.

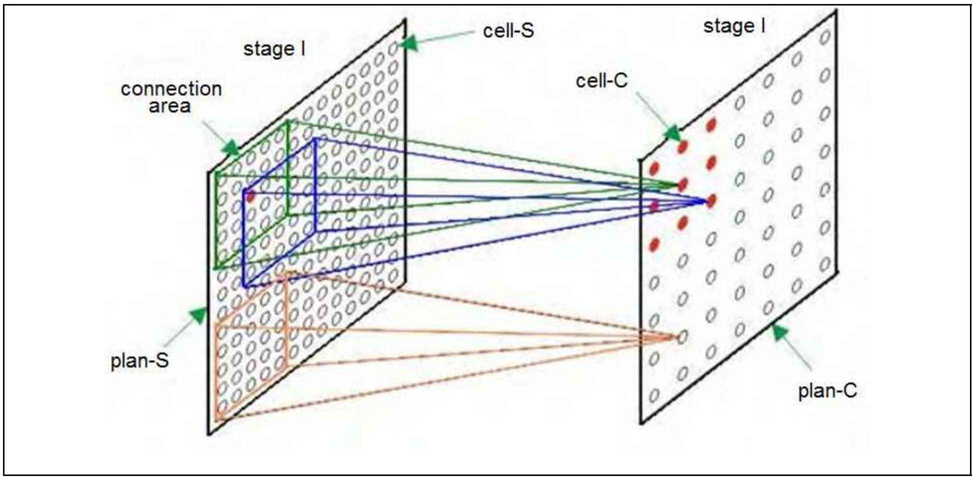

Fig. 10. Example of the connection area of a Cell-C.

3.4 Network Training

Although there are two main training methods for the Neocognitron network, it is described here the method originally designed, which is learning without supervision.

At first, the training follows as the majority of neural networks, i.e., it is showed a sample pattern, and data are propagated through the network, allowing the weights of the connections to fit progressively according to a given algorithm. After the weights are updated, the network receives a second pattern in the input layer, and the process repeats with all the training samples until the network classifies the patterns correctly.

Neocognitron network has the characteristic that all the cells in the same cell-plan share the same set of weights. Therefore, only a single cell of each plan must participate in training, and after that, distribute the whole weight to the other cells.

To better understand the operation, one can imagine all plans of a Layer-S stacked on each other, aligned so that the cells corresponding to a given location is directly above each other. Thus, it is possible to imagine several columns, cutting perpendicularly the planes. These columns create groups of cells-S, where all group members have receptive fields in the same location in the input layer.

With this model in mind, we can now apply a standard input and examine the response of Cell-Ss in each column. To ensure that each Cell-S provides a distinct response, one may start a_l weights with random small positive value and the weights b_l inhibitors with zero. First, note the plane and the position of the Cell-S whose response is the strongest in each column. Then it examines the plans individually so that if a plan has two or more of these Cell-Ss, it chooses only the Cell-S with the stronger response, subject to the condition that each cell is in a different column-S.

These Cell-Ss become the prototypes or representatives of all the cells in the respective plan. Once chosen the representatives, the updates of the weights are made in accordance with the Equation 4 and Equation 5, and all the cells of the same plan will be updated to be with the same weights:

$$\Delta a_l(k_{l-1}, v, k_l) = q_l c_l(u) u_{cl-1}(k_{l-1}, n+v) \tag{4}$$

$$\Delta b_l(k_l) = q_l v_{cl}(n) \tag{5}$$

Once the cells are updated to respond to a particular characteristic, they begin to emit responses smaller in relation to other features.

4. GPU as a Device to Generic Processing

Over the past 10 years, hitherto, it has seen the evolution of the GPU's as specialized hardware to process graphics and video output, and massive parallel processing of data for general computing. The power of data processing of GPU's has grown much faster than the CPU, and the main reason for this rapid growth of GPU's with respect to the CPU is due to the fact that the GPU's were born with the focus of intensive computing, with respect to data processing and massive parallel computing, as just the minimum requirements necessary to meet the needs of the scenario of computer graphics, like rendering, shadows in 3D scenes and others.
Thus the design of the GPU takes into account the existence of more transistors dedicated to a better process control and data flow, as illustrated schematically in Figure 12, which depicts the main elements: ALU, cache, and DRAM control for a CPU (Figure 12a) and a GPU (Figure 12b).

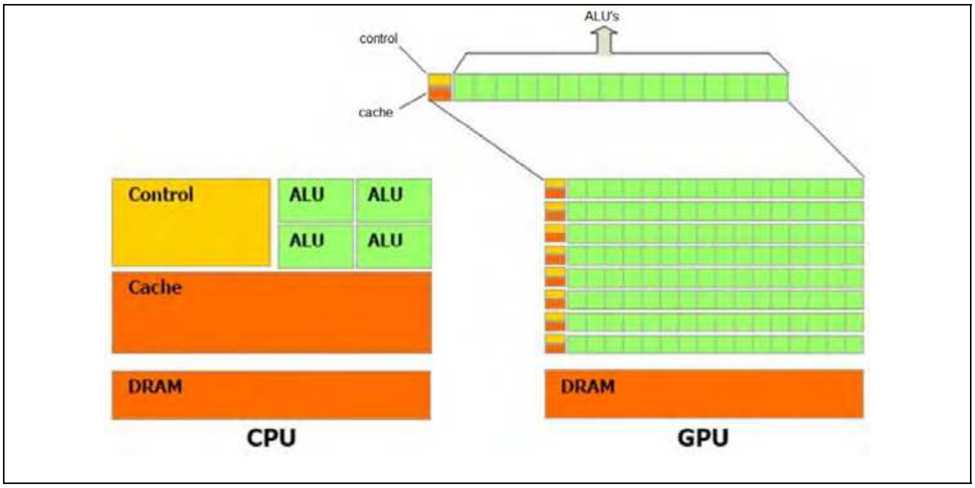

Fig. 11. GPU intended to use more transistors for Data Processing.

Many applications that process large data sets organized in a matrix/vector can use a model of parallel computing. In 3D rendering processes large arrays of pixels and vertices are organized so that they can be processed in parallel using threads. Similarly, applications of image processing, encoding and decoding, video scaling, stereo vision, artificial neural networks and pattern recognition can be processed in data blocks and pixels by parallel threads. In fact, many algorithms, even outside the area of image processing, can be accelerated through parallelization of data processing, specially signal processing, simulation of physical effects, computer models of financial or biological applications.

4.1 CUDA - Compute Unified Device Architecture

The development of applications that use the GPU as a device for "unconventional" parallel data processing, i.e., not specifically the graphics processing like rendering, is increasing. However the use of a GPU as a device that requires an adjustment of the traditional graphics card pipeline's, forcing the developer to take responsibility for certain control points in these processes, through graphics libraries that have an API for GPU's to become programmable, is annoying.

CUDA is a new architecture of hardware and software that was developed with the main objective of managing the parallel processing of data within the GPU device without the need to make the mapping of the routines and take responsibility for the execution of the pipeline system, through API chart.

In Figure 12 we have the software stack environment of CUDA, not necessarily for 4 layers of software and these are: (a) application, which is implemented by the browser software that makes use of GPU as a device data processing; (b) CUDA Library is a set of mathematical libraries, such as CUBLAS, an extension of a BLAS library functions algebra implemented in FORTRAN and CUFFT's a fast Fourier transform of 1, 2 and 3 dimensions; (c) where the CUDA runtime routines of other graphics libraries like OpenGL and DirectX are accessed to be processed on the GPU; and (d) CUDA Driver API that is the direct communication with the GPU.

In order to facilitate the development of computing solutions for general purpose, not just graphic, CUDA provides the GPU direct memory access to both writing (Figure 13) and for reading (Figure 14), just as a conventional CPU works.

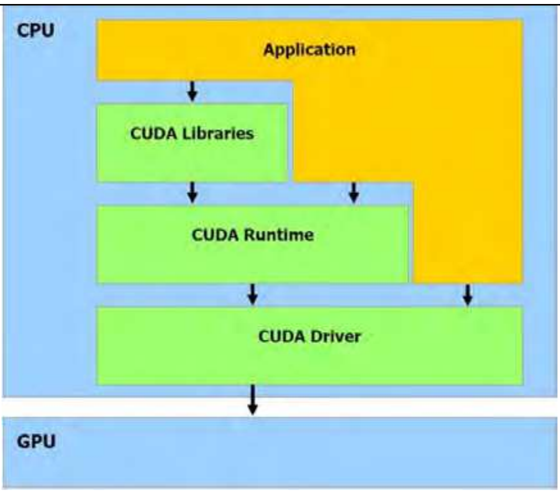

Fig. 12. CUDA Software Stack (NVIDIA, 2007).

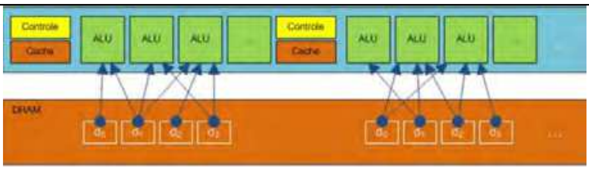

Fig. 13. GPU accessing memory to read (NVIDIA, 2007).

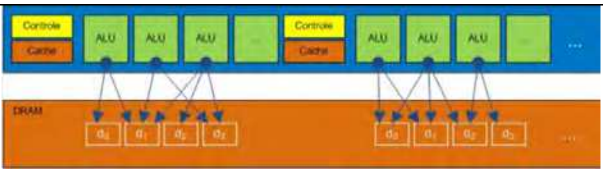

Fig. 14. GPU accessing memory to write (NVIDIA, 2007).

In these Figures 14 and 15, the data is read from or written to memory by the ALUs. In this architecture there is a parallel data cache and a shared memory, which has a high-speed access for both, writing and reading. The applications benefit from this structure by minimizing overfetch and round-trips of DRAM and reduce the need/dependence on the bandwidth of DRAM access.

4.2 CUDA Programming Model

In developing a parallel application via CUDA, GPU is viewed as a computer device capable of running a large number of threads in parallel. The GPU operates as a coprocessor of the CPU, which in the context of CUDA is called the host.

The part of the application, most suitable to be processed in the device, is a function performed several times with different data. These functions should be isolated and implemented within the scope of CUDA and are called the kernel that are executed within the device.

Both host and device (GPU) have one call to a DRAM memory device and a host memory. The call is made from a kernel due to the transfer of data between two memories. CUDA provides a set of functions for this feature (moving data between the two types of memory).

When a host application makes a call to a kernel, it is executed by a set of threads arranged in blocks of execution. These blocks in turn are grouped into grid blocks.

A block of threads is a lot of threads that work together cooperatively to get a better efficiency of data usage and shared memories, and their processing is synchronized. Each thread within a block is identified by a threadID, which is a combination of the number of thread with the block in which it is inserted.

The formation of a value of one threadID is complex, and to assist in this process, it can be specified a block to have two or three dimensions of arbitrary size, and identify each thread using a composite index of two or three instances, as shown in Table 1, where Dx, Dy, Dz are dimensions of the blocks, x, y, z are the coordinates, and threadID is obtained by calculating the expressions presented.

Block Size	Coordinated Thread	threadID
D_x, D_y	x,y	$x+yD_x$
D_x, D_y, D_z	x,y,z	$x+yD_x+zD_xD_y$

Table 1. Formation of the address of the thread within the block

The number of threads that a block can contain is limited. As previously mentioned, blocks with the same dimensionality working in the execution of a single kernel can be grouped into a grid of blocks of threads. The call of this kernel is performed using a specific syntax which is reported beyond the normal parameters of the function to be processed on the device: data on grid (Dg), block (Db) and memory to be allocated (Ns).
As the threads, blocks also have an identification number within a grid, following a rule similar to the formation of the address of the threads, as shown in Table 2, where Dx, Dy indicate the dimensions of the grid, x, y the coordinates of blockID blocks and the identification number of the block calculated by the showed expression.

Grid Size	Coordinated Block	blockID
D_x, D_y	x,y	$x+yD_x$

Table 2. Formation of the address of the block within the grid

In Figure 15 presents an overview of the structure of the threads running inside the device. This can be seen separating the two: host hardware (CPU) and the device (GPU), where the kernels called for implementation on the host are sent to the device, where the processing of threads arranged in blocks are divided into grids of processing.
It is worth calling attention here to the fact that kernels have distinct grid settings, and different blocks, as its dimensionality, as shown in Figure 15, where the size of the blocks and the grid used in kernel 2 is different from that used by the kernel 1.

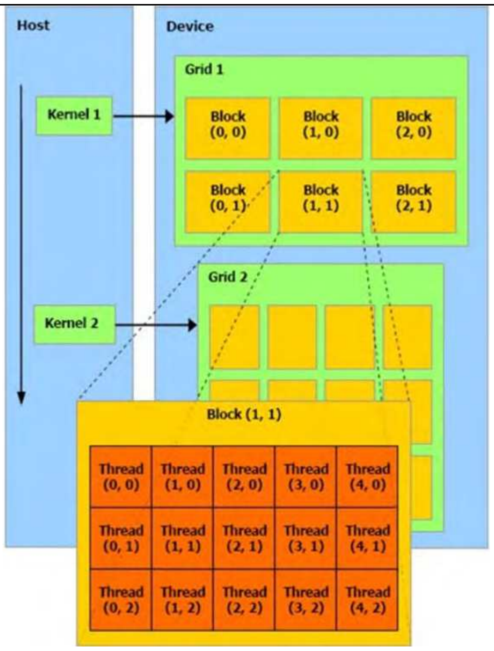

Fig. 15. Address of the threads within the blocks to the grid (NVIDIA, 2007).

4.3 CUDA Memory Model

The thread runs inside the device and only has (DRAM) memory access inside this, according to a set of access rules shown in Figure 16 and detailed in Table 3.

The threads can access registers (register) and memory space for reading and writing. The shared memory (shared) is accessed by blocks for writing and reading. The global memory is accessed by grid for reading and writing. Memories of constant and textures are accessed by the grid in read-only.

The global spaces, constant, and texture can be read or written by the host and are persistent across kernel calls during the same application.

Memory Space	When accessed	Rule
Register	by thread	Read/Write
Local	by thread	Read/Write
Shared	by block	Read/Write
Global	by grid	Read/Write
Constant	by grid	Read Only
Texture	by grid	Read Only

Table 3. Memory Model Access Rules

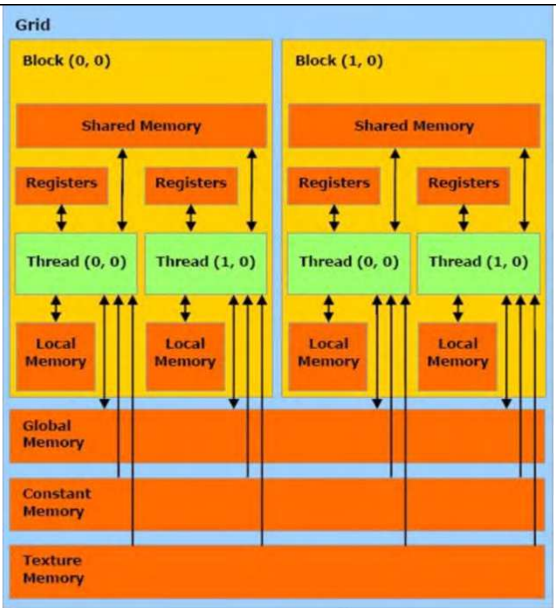

Fig. 16. CUDA Memory Model (NVIDIA, 2007).

5. Processing Neocognitron Network with CUDA

A face recognition system using Neocognitron neural network can be processed in two phases: the learning phase, and the recognition phase. This work have focused on recognition, since, the learning phase is ready. The training is being carried out by an application developed in Delphi (Saito and Abib, 2005) and it has as a product of its execution, the generation of a repository of data. This comprises a set of three types of binary files: one to store the number of plans for each stage, another type to store the weight-a e weight-b trained by the network and a third type with the results used by the layer of Cell-Cs of the stage.
At the recognition phase, one face image (input pattern) is shown to the system, and it executes and tries to identify the face. Figure 17 shows the block diagram of the parallel processing management algorithm of the face recognition phase using CUDA, in two parts, host and device.

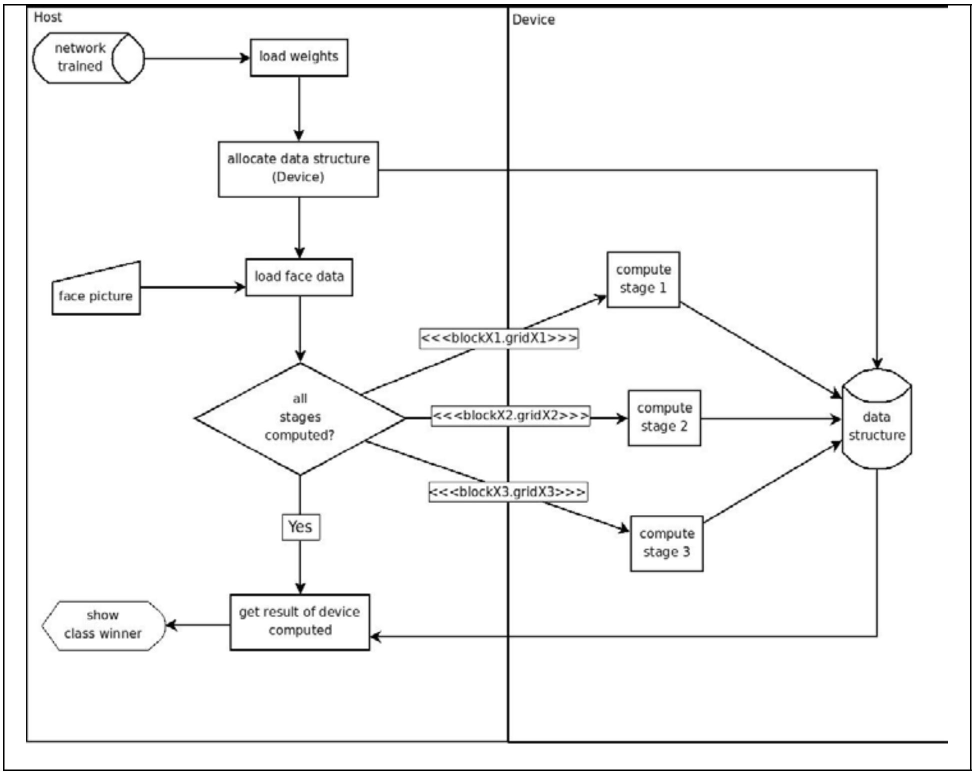

Fig. 17. Block Diagram Organization of Processing Neocognitron on CUDA device.

As can be seen in the block diagram (Figure 17), there are two repositories of data, one with the weights produced during the training phase of the network, and the second repository of data with the images of the faces to be recognized. We used two banks of the faces, one developed at UFSCar (Figure 19), consisting of fifty frontal images of six persons, in 57x57 pixels resolution; and the CMU PIE database (Figure 20) which consists of a large number of images of people in different poses, lighting and facial expressions. It used 13 cameras, 9 in the same horizontal line, each separated from 22.5^0. Other 4 cameras include 2 above and below the central camera, and 2 in the corners of the room. On Figure 19, the different positions of cameras are identified by ratings c02 ... c34. To obtain the change in lighting, it used a system with 21 flashes. Capturing images with and without the backlight, are obtained 43 different lighting conditions. For a variety of facial expressions were asked for people to neutral expressions, smile, blinking, and speaking. The database consists of 41368 images of 68 people.

Fig. 18. UFSCar Face Data Base.

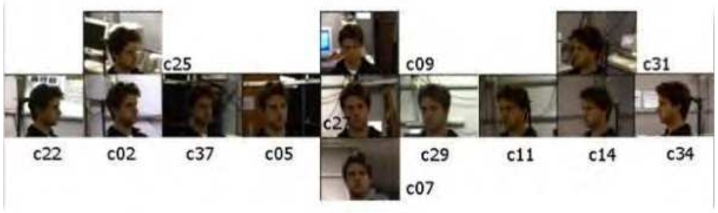

Fig. 19. CMU PIE database.

Since this work corresponds to the recognition phase, they were selected randomly 10 people of the database CMU PIE (4002, 4014, 4036, 4047, 4048, 4052, 4057, 4062, 4063, e 4067). They were selected the images of people speaking, because the existence of 60 images per pose, per person. Thus, during the experiments, they were used frontal images of people with size 640x486, and after the capture of the face region, the reduced image of size 57x57, as shown in Fig. 21.

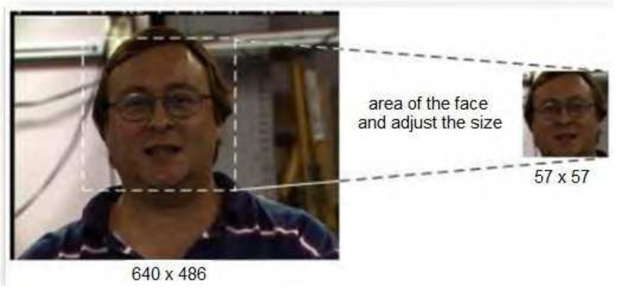

Fig. 20. Face picture used to recognition process.

At Figure 21, it can be seen the Neocognitron network processing, to the process of recognition. It may be noted the three stages of the network, represented by: stage 1, formed by all the layers U_{S1}, U_{C1} and U_{V1}; stage-2, formed by all the layers U_{S2}, U_{C2} and U_{V2} and stage-3, formed by all the layers U_{S3}, U_{C3} and U_{V3}. Also, it is presented the input layer U_0.

Table 4 shows the dimensionality of the weights used in the network, according to the stage where it is applied. The weight-a and weight-b are obtained by the training process of the network, which in the scope of this project is already done.

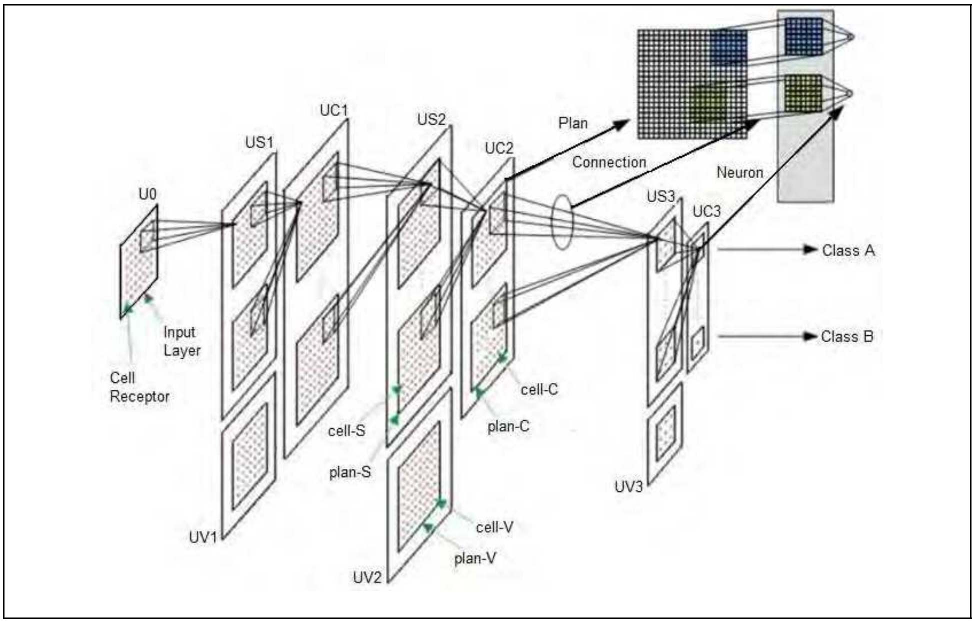

Fig. 21. Neocognitron network processing using GPU.

The weight-c and weight-d are fixed and defined at the time of implementing the network. On the Table 5, 6 and 7 are presented as matrices weight-c used in stages 1, 2 and 3 respectively.

Stage	Weight-a	Weight-b	Weight-c	Weight-d
1	7x7	1x1	7x7	5x5
2	7x7	1x1	7x7	5x5
3	5x5	1x1	5x5	3x3

Table 4. Dimensionality of the Weights used in the network.

0.017361	0.019231	0.019231	0.019231	0.019231	0.019231	0.017361
0.019231	0.019231	0.021635	0.021635	0.021635	0.019231	0.019231
0.019231	0.021635	0.021635	0.024038	0.021635	0.021635	0.019231
0.019231	0.021635	0.024038	0.026709	0.024038	0.021635	0.019231
0.019231	0.021635	0.021635	0.024038	0.021635	0.021635	0.019231
0.019231	0.019231	0.021635	0.021635	0.021635	0.019231	0.019231
0.017361	0.019231	0.019231	0.019231	0.019231	0.019231	0.017361

Table 5. Matrix of weight-c of stage 1.

0.017361	0.019231	0.019231	0.019231	0.019231	0.019231	0.017361
0.019231	0.019231	0.021635	0.021635	0.021635	0.019231	0.019231
0.019231	0.021635	0.021635	0.024038	0.021635	0.021635	0.019231
0.019231	0.021635	0.024038	0.026709	0.024038	0.021635	0.019231
0.019231	0.021635	0.021635	0.024038	0.021635	0.021635	0.019231
0.019231	0.019231	0.021635	0.021635	0.021635	0.019231	0.019231
0.017361	0.019231	0.019231	0.019231	0.019231	0.019231	0.017361

Table 6. Matrix of weight-c of stage 1.

0.035225	0.039628	0.039628	0.039628	0.035225
0.039628	0.039628	0.044031	0.039628	0.039628
0.039628	0.044031	0.048924	0.044031	0.039628
0.039628	0.039628	0.044031	0.039628	0.039628
0.035225	0.039628	0.039628	0.039628	0.035225

Table 7. Matrix of weight-c of stage 1.

Tables 8, 9 and 10 corresponds to the matrices of weight-d used in the processing of stages 1, 2, and 3, respectively.

0.72	0.72	0.72	0.72	0.72
0.72	0.81	0.9	0.81	0.72
0.72	0.9	1	0.9	0.72
0.72	0.81	0.9	0.81	0.72
0.72	0.72	0.72	0.72	0.72

Table 8. Matrix of weight-d of stage 1.

0.72	0.81	0.81	0.81	0.72
0.81	0.81	0.9	0.81	0.81
0.81	0.9	1	0.9	0.81
0.81	0.81	0.9	0.81	0.81
0.72	0.81	0.81	0.81	0.72

Table 9. Matrix of weight-d of stage 2.

0.81	0.9	0.81
0.9	1	0.9
0.81	0.9	0.81

Table 10. Matrix of weight-d of stage 3.

To carry out processing of any type of data within the CUDA, it must be able to be implemented within a hierarchical organization such as:

Grid >> Block >> Thread

meaning that the grid are composed by blocks, and the blocks by threads. The Neocognitron network also has an organization in a hierarchical structure, such as:

Stage >> Cell Plan >> Neuron

meaning that the stages are composed by several call-plans, and the cell-plans of a collection of cells, or neurons.
Analyzing the organization of the two architectures, it is possible to verify some points of correspondence, which can be seen in Figure 21 and listed in Table 11, which sees itself as a Grid equivalent to a Stage the block equivalent to all the neuron processing, and the thread equivalent to one connection processing. The correspondence between the two architectures facilitates the network modelling to be used at the CUDA/GPU environment.

CUDA	Neocognitron
Grid	Stage
Block	Cell Plan
Thread	Neuron

Table 11. Points of correspondence: CUDA x Neocognitron.

Another important factor of the validity of the correspondence between the architectures lies on the fact that there is an independence of the values of a huge amount of neurons at their processing. That is, the value of a neuron, in a cell-plan does not depend on the value of the neighbour neuron at the same plane, but on the data of the preceding stage. That validates the use of architecture as the GPU/CUDA.
The implementation of a project using the GPU/CUDA, determines that there are two processing environments, the host and device. It was developed a set of functions (processes) that run in the host and only a function that is performed on the device, as can be seen in Figure 17.
This kernel has the responsibility to process a single stage of the Neocognitron network, and is called by the host application in each stage within a specific order. It should be noted that the network model in this project has three stages.
Despite the processing of the network be similar for all stages, the Neocognitron network shows a reduction of dimensionality during its processing. The plan size of each stage is reduced from stage to stage, until the last stage, which has a single neuron in a plan, and a number of plans coincident to the number of classes to be recognized.
This is why the kernel is required at the time of processing a certain stage and can thus tell the GPU setting specific with respect to size of blocks of processing to be implemented. The models of cell-plans and connection area organizations, invoked by the kernels, by stage, are shown in Table 12. The goal is to process a plan with the greatest number of areas of possible connections.

Stage	Plan-S	Con. Area	Plan-C	Block	Grid
1	21x21	7x7	21x21	5x49	1
2	14x14	7x7	14x14	4x49	1
3	7x7	5x5	1x1	10x25	1

Table 12. Organization of cell-plans and connection area and its implementation in GPU/CUDA.

As a block processed in a single cycle of GPU/CUDA, the data of 16 multiprocessors has been that for the Stage-1. Each plan has a size of 21x21 (441) neurons and the connection area of this plan has a dimension of 7x7, resulting in 49 connections. The size of the block used for the processing of this stage was 49 threads, the same size of the connection area, 5 blocks in the grid processing simultaneously, or 245 connections computed simultaneously, as can be seen in Figure 22.

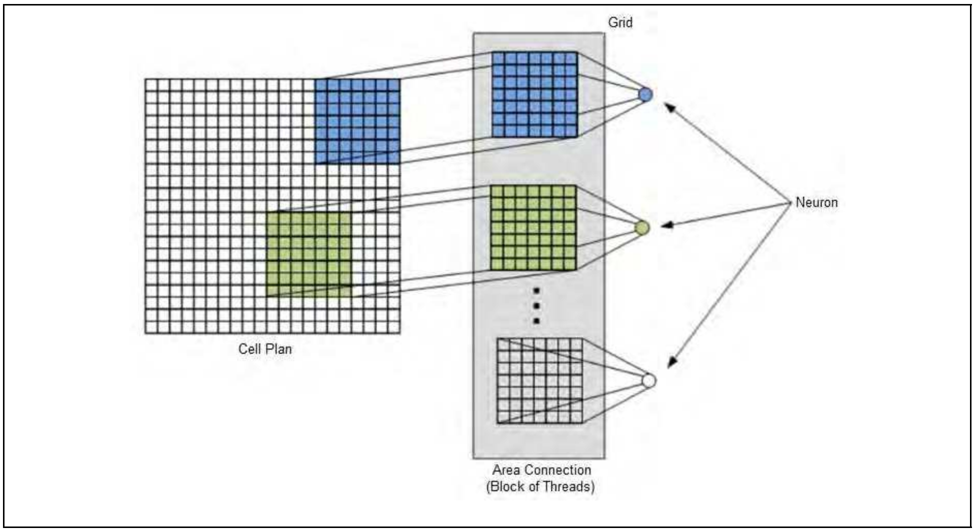

Fig. 22. Neuron connection processing diagram.

The total number of neurons processed simultaneously from 245, 245 and 250, to the stages 1, 2 and 3 respectively.

6. Results

Using the UFSCar and CMU-PIE human face databases, the recognition rate obtained is high and probably may be increased using more training images. The results of the recognition rate obtained by the two databases can be seen in Table 13.

Database Face	Rate Recognition
CMU-PIE	98%
UFSCar	97%

Table 13. Degree of accuracy of recognition of faces.

The total run time of the network was 0.118 seconds. The measure of time was obtained through the use of the control functions of processing time made available by the API CUDA, and used the cutCreateTimer functions, cutStartTimer, cutStopTimer, cutGetTimerValue and cutDeleteTimer.

Stage	Plan-S	time (sec)
1	95 plans	0.092
2	51 plans	0.022
3	47 plans	0.004

Table 14. Time, in seconds, spent during cell-plan processings.

Table 15 presents a comparison between the processing time in the GPU / CUDA, with the same network being processed in a single (mono) environment and processed in a cluster with 8 processors, values obtained by Ribeiro (Ribeiro 2002). It was made an adjustment in time goals for the work of Ribeiro, depending on the speed of processors used in their work and the existing today. The table is organized into three columns, where the first column is the computer architecture, the second column is the number of parallel processors, and the third column, processing time in seconds.

Architecture	Number of Processors	Time (sec)
Mono	1	48
Cluster	8	15
GPU/CUDA	128	0.118

Table 15. Comparing the processing time in different architectures.

By this table it is possible to calculate the speed-up and efficiency of processing between the architectures. These values are presented in Table 16, organized into three columns: computer architecture, Speed-up, and efficiency.

Architecture	Speed-up	Efficiency
Cluster	79.787	0.311
GPU/CUDA	255.319	0.999

Table 16. Comparation Speed-up and Efficiency in different architectures.

The total amount of memory used at the device was 439 MB, which represents an allocation of 57% of total memory. Since this is their distribution and consumption detailed at Table 17, in two columns, the first "Reserve Area" indicates where it allocated the amount of memory in Mega Bytes presented in the second column "Located Area".

Reserved Area	Located Area (Mb)
Stage 1	336
Stage 2	80
Stage 3	10
Other Variables	13

Table 17. Number of dedicated memory, of GPU, allocated for the implementation.

7. Conclusion

With the Neocognitron network processing within the GPU/CUDA presented in this work, we can conclude that there was a significant increase on the processing performance of the Neocognitron face recognition, showing the feasibility of using this method.
However the size of images used in the operation were small, 57 x 57, which allowed the full load of the structure of the network into the memory of the device where access is protected and high-speed, factors that may have influenced the results presented.
In an attempt to draw a line between the comparative Neocognitron network processed in the GPU/CUDA and traditional architecture it was verified through the calculation of speed-up, a gap, since they won a super-linear speed-up $S_p > p$. This occurred by differences in architecture. Moreover a high performance was observed when compared the time of processing.
Another conclusion on the implementation of this project is that this minimizes some existing common problems, when used other parallel computing environment, cluster that for example, you can quote:

- Synchronization: since the granularity of development within this device is not competing for shared memories (each thread has its point / area of memory) there is a need for loss of time for achieving a synchronization of processors;
- Network: if the whole process takes place within the same device there is the issue connection type and the speed of the entire GPU architecture; and
- Contention: there is no competition for resources by processors.

Another issue, where the GPU has advantages over the traditional architecture of high performance computing (such as cluster), is related to load balancing. Since the GPU architecture is focused on SIMD data processing type, the Neocognitron network implementation project, focused on block processing, is privileged, since an entire block is processed in a single cycle of processing.
However, the development of projects in GPU/CUDA environment presents as the main difficulty the modelling of streaming data processing. As seen in other studies by Poli (Poli et al. 2007) (Poli et al. 2008), it is not every applications that benefit with this architecture. It can be submitted three categories of possibilities for implementation of applications in the GPU: full potential for development, partial potential for development and the unfeasible development.
The applications that benefit with the processing in GPU/CUDA, are those that have large volumes of data on a matrix, and have their processing independent of the adjacent processing. The computer cost for decision making is significant. It is concluded that the shorter the granularity inside a model of data organized and structured to a vision processing in blocks will have a better gain in processing performance.

8. Acknowledgents

The authors would like to thank Prof. Simon Backer from Carnegie Mellon University for the kindness of sending the PIE database, used in this work.

9. References

Fukushima K. and Miyake S. (1982). *Neocognitron: A New Algorithm for Pattern Recognition Tolerant of Deformations and Shift in Position*, Pattern Recognition, Vol. 15, pages 455-469

Fukushima K. and Wake N. (1992). *Improved Neocognitron with Bend-Detection Cells*, IEEE - International Joint Conference on Neural Networks, Baltimore, Maryland, 1992

NVIDIA (2007). *NVIDIA CUDA Compute Unified Device Architecture - Programming Guide,* Publisher NVIDIA

Poli G., Levada A. M. L., Mari J. F., Saito J. H. (2007). *Voice Command Recognition with Dynamic Time Warping (DTW) using Graphics Processing Units (GPU) with Compute Unified Device Architecture (CUDA)*, SBAC-PAD International Symposium on Computer Architecture and High Performance Computing, 2007, pages 19-27

Poli G., Levada A. M.L., Saito J. H. , Mari J. F. , Zorzan M. R. (2008). *Processing Neocognitron of Face Recognition on High Performance Environment Based on GPU with CUDA Architecture (CUDA)*, SBAC-PAD International Symposium on Computer Architecture and High Performance Computing, 2008, pages 81-88

Saito J. H. and Fukushima K. (1998). *Modular Structure of Neocognitron to Pattern Recognition*, ICONIP'98, Fifth Int. Conf. on Neural Information Processing, Kitakyshu – Japan 1998

Hirakuri M. H. (2003). *Aplicação de Rede Neural Neocognitron para reconhecimenro de atributos faciais*, Dissertação de Mestrado - Universidade Federal de São Carlos, 2003

Ribeiro, L. J. (2002). *Paralelização da Rede Neural Neocognitron em Cluster SMPs.*, Dissertação de Mestrado da Universidade Federal de São Carlos (UFSCar), 2002

Sim T. , Bsat M. (2003). *The CMU Pose, Illumination, and Expression database*, IEEE Transaction on Pattern Analysis and and Machine Intelligence, 2003, pages 1615-1618

Saito J. H. and Abib S. (2005). *Using CMU PIE Human face database to a Convolutional Neural Network - Neocognitron*, ESANN2005 - European Symposium on Artificial Neural Network, 2005, pages 491-496

Terra Notícias (2006) *China implanta sistemas de reconhecimento facial biométrico*, http://noticias.terra.com.br/ciencia/interna/0,,OI956783-EI238,00.html

Face Recognition
Edited by Milos Oravec

ISBN 978-953-307-060-5
Hard cover, 404 pages
Publisher InTech
Published online 01, April, 2010
Published in print edition April, 2010

This book aims to bring together selected recent advances, applications and original results in the area of biometric face recognition. They can be useful for researchers, engineers, graduate and postgraduate students, experts in this area and hopefully also for people interested generally in computer science, security, machine learning and artificial intelligence. Various methods, approaches and algorithms for recognition of human faces are used by authors of the chapters of this book, e.g. PCA, LDA, artificial neural networks, wavelets, curvelets, kernel methods, Gabor filters, active appearance models, 2D and 3D representations, optical correlation, hidden Markov models and others. Also a broad range of problems is covered: feature extraction and dimensionality reduction (chapters 1-4), 2D face recognition from the point of view of full system proposal (chapters 5-10), illumination and pose problems (chapters 11-13), eye movement (chapter 14), 3D face recognition (chapters 15-19) and hardware issues (chapters 19-20).