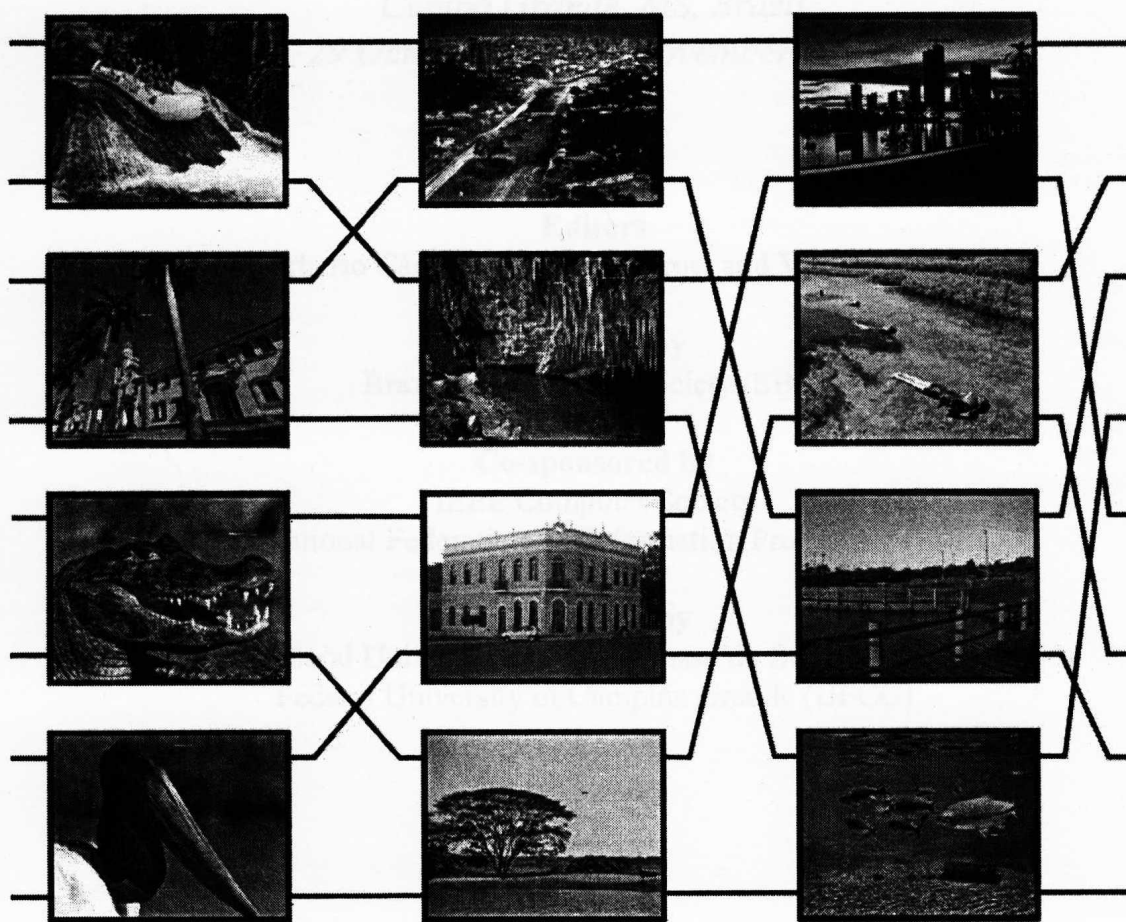


Proceedings

20th International Symposium on Computer Architecture and High Performance Computing



Proceedings

20th International Symposium on Computer Architecture and High Performance Computing

Campo Grande, MS, Brazil
29 October 2008 - 1 November 2008

Editors

Edson Norberto Cáceres, Walfredo Cirne, and Viktor K. Prasanna

Promoted by

Brazilian Computer Society (SBC)

Co-sponsored by

IEEE Computer Society

International Federation for Information Processing (IFIP)

Organized by

Federal University of Mato Grosso do Sul (UFMS)

Federal University of Campina Grande (UFCG)



Los Alamitos, California
Washington • Tokyo



QA16.1
.A73595
2008

Copyright © 2008 by The Institute of Electrical and Electronics Engineers, Inc.

All rights reserved.

Copyright and Reprint Permissions: Abstracting is permitted with credit to the source. Libraries may photocopy beyond the limits of US copyright law, for private use of patrons, those articles in this volume that carry a code at the bottom of the first page, provided that the per-copy fee indicated in the code is paid through the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923.

Other copying, reprint, or republication requests should be addressed to: IEEE Copyrights Manager, IEEE Service Center, 445 Hoes Lane, P.O. Box 133, Piscataway, NJ 08855-1331.

The papers in this book comprise the proceedings of the meeting mentioned on the cover and title page. They reflect the authors' opinions and, in the interests of timely dissemination, are published as presented and without change. Their inclusion in this publication does not necessarily constitute endorsement by the editors, the IEEE Computer Society, or the Institute of Electrical and Electronics Engineers, Inc.

IEEE Computer Society Order Number P3423
BMS Part Number CFP08307-PRT
ISBN 978-0-7695-3423-7
ISSN Number 1550-6533

Additional copies may be ordered from:

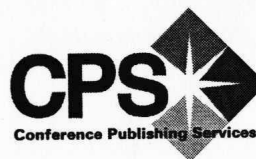
IEEE Computer Society
Customer Service Center
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1314
Tel: +1 800 272 6657
Fax: +1 714 821 4641
<http://computer.org/cspress>
csbooks@computer.org

IEEE Service Center
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
Tel: +1 732 981 0060
Fax: +1 732 981 9667
[http://shop.ieee.org/store/
customer-service@ieee.org](http://shop.ieee.org/store/customer-service@ieee.org)

IEEE Computer Society
Asia/Pacific Office
Watanabe Bldg., 1-4-2
Minami-Aoyama
Minato-ku, Tokyo 107-0062
JAPAN
Tel: +81 3 3408 3118
Fax: +81 3 3408 3553
tokyo.ofc@computer.org

Individual paper REPRINTS may be ordered at: <reprints@computer.org>

Editorial production by Lisa O'Conner
Cover art production by Joe Daigle/Studio Productions
Printed in the United States of America by The Printing House



**IEEE Computer Society
Conference Publishing Services (CPS)**

<http://www.computer.org/cps>

Table of Contents

20th International Symposium on Computer Architecture and High Performance Computing

SBAC-PAD

Message from the General Chair	viii
Message from the Program Committee Chairs	ix
Conference Organizers	x
Program Committee	xi
Reviewers	xiii
Brazilian Computer Society	xiv

Session 1: Architecture I

Accurate and Low-Overhead Dynamic Detection and Prediction of Program Phases Using Branch Signatures	3
<i>Balaji Vijayn and Dmitry V. Ponomarev</i>	
Aggressive Scheduling and Speculation in Multithreaded Architectures: Is it Worth its Salt?	11
<i>Jason Loew and Dmitry Ponomarev</i>	
An Optimization Mechanism Intended for Two-Level Cache Hierarchy to Improve Energy and Performance Using the NSGAII Algorithm	19
<i>Abel G. Silva-Filho, Carmelo J.A. Bastos-Filho, Davi M.A. Falcão, Filipe R. Cordeiro, and Rodrigo M.C.S. Castro</i>	

Session 2: Applications I

On Simulated Annealing for the Scheduling of Parallel Applications	29
<i>Rodrigo Fernandes de Mello and Luciano José Senger</i>	
Controlling Processes Reassignment in BSP Applications	37
<i>Rodrigo da Rosa Righi, Laércio Lima Pilla, Alexandre Carissimi, and Philippe O.A. Navaux</i>	
A High Performance Massively Parallel Approach for Real Time Deformable Body Physics Simulation	45
<i>Thiago S.M.C de Farias, Mozart W.S. Almeida, João Marcelo X.N. Teixeira, Veronica Teichrieb, and Judith Kelner</i>	

Session 3: Multicore

A Methodology for Developing High Fidelity Communication Models for Large-Scale Applications Targeted on Multicore Systems	55
<i>Charles W. Lively, Valerie E. Taylor, Sadaf R. Alam, and Jeffrey S. Vetter</i>	
Selection of the Register File Size and the Resource Allocation Policy on SMT Processors	63
<i>Jesús Alastruey, Teresa Monreal, Francisco Cazorla, Victor Viñals, and Mateo Valero</i>	
ORBIT: Effective Issue Queue Soft-Error Vulnerability Mitigation on Simultaneous Multithreaded Architectures Using Operand Readiness-Based Instruction Dispatch	71
<i>Xin Fu, Tao Li and José Fortes</i>	

Session 4: Applications II

Processing Neocognitron of Face Recognition on High Performance Environment Based on GPU with CUDA Architecture	81
<i>Gustavo Poli, José Hiroki Saito, João F. Mari, and Marcelo R. Zorzan</i>	
Parallel Verified Linear System Solver for Uncertain Input Data	89
<i>Mariana Kolberg, Márcio Dorn, Luiz Gustavo Fernandes, and Gerd Bohlender</i>	
Applying Virtualization and System Management in a Cluster to Implement an Automated Emulation Testbed for Grid Applications	97
<i>Rodrigo N. Calheiros, Mauro Storch, Everton Alexandre, César A.F. De Rose, and Marcus Breda</i>	

Session 5: Architecture II

Hiding Communication Delays in Clustered Microarchitectures	107
<i>Robert J. LaDuca, Joseph Sharkey, and Dmitry V. Ponomarev</i>	
Software Synthesis for Hard Real-Time Embedded Systems with Energy Constraints	115
<i>Eduardo Tavares, Bruno Silva, Paulo Maciel, and Pedro Dallegrave</i>	
A Segmented Bloom Filter Algorithm for Efficient Predictors	123
<i>M. Breternitz, Gabriel H. Loh, Bryan Black, Jeffrey Rupley, Peter G. Sassone, Wesley Attrot, and Youfeng Wu</i>	

Session 6: Grid, Cluster, and Operating Systems

Measuring Operating System Overhead on CMT Processors	133
<i>Petar Radojkovic, Vladimir Cakarevic, Javier Verdú, Alejandro Pajuelo, Roberto Gioiosa, Francisco J. Cazorla, Mario Nemirovsky, and Mateo Valero</i>	
Aspect-Based Patterns for Grid Programming	141
<i>Luis Daniel Benavides Navarro, Rémi Douence, Fabien Hermenier, Jean-Marc Menaud, and Mario Südholt</i>	
A Reconfigurable Run-Time System for Filter-Stream Applications	149
<i>Daniel Fireman, George Teodoro, André Cardoso, and Renato Ferreira</i>	

Session 7: Memory Systems

Using Analytical Models to Efficiently Explore Hardware Transactional Memory and Multi-Core Co-Design _____	159
<i>James Poe, Chang-Burm Cho, and Tao Li</i>	
Performance Sensitivity of NUCA Caches to On-Chip Network Parameters _____	167
<i>Alessandro Bardine, Manuel Comparetti, Pierfrancesco Foglia, Giacomo Gabrielli, and Cosimo A. Prete</i>	
A Software Transactional Memory System for an Asymmetric Processor Architecture _____	175
<i>Felipe Goldstein, Alexandro Baldassin, Paulo Centoducatte, Rodolfo Azevedo, and Leonardo A.G. Garcia</i>	
Transactional WaveCache: Towards Speculative and Out-of-Order DataFlow Execution of Memory Operations _____	183
<i>Leandro A.J. Marzulo, Felipe M.G. Franca, and Vitor Santos Costa</i>	
Author Index _____	191

Processing Neocognitron of Face Recognition on High Performance Environment Based on GPU with CUDA Architecture

Gustavo Poli, José Hiroki Saito, João F. Mari, Marcelo R. Zorzan
Universidade Federal de São Carlos
Computer Department
São Carlos - SP - Brazil
{gustavo_silva,saito,joao_mari,marcelo_zorzan}@dc.ufscar.br

Abstract

This work presents an implementation of 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 planes. 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 UFS-Car, and the CMU-PIE (Carnegie Mellon University Pose, Illumination and Expression) database. The load balancing was achieved through the use of cellular connections as threads organized in blocks, following the CUDA philosophy of development. The results showed the feasibility of this type of device as a massively parallel data processing tool, and that smaller the granularity and the data dependency of the parallel processing, better is its performance.

1. Introduction

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 finalities. On China, for example, it was developed

the immigrant recognition system to the cities of Shenzhen and Zhuhai [4]. 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 basic variables of the process: the degree of reliability/robustness of the technique being used and the computational cost of this technique.

The goal of this paper is the study of a computer architecture for face recognition, aiming the 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 generic the CUDA (Compute Unified Device Architecture), a library that extends the functions of language C in order to provide the GPU as a device for data processing [5].

This section is followed by the section 2, which describes the neocognitron neural network; section 3 describes CUDA, as the architecture for the use of GPU; section 4 describes the methodology used to the face recognition implementation; section 5 presents the results, and section 6, conclusions.

2. The Neocognitron Structure

Neocognitron is a massively parallel neural network, composed by several layers of neuron cells, proposed by Fukushima [1] [2] [9]. In a brainway computer it corresponds to part of the human visual recognition system.

Neocognitron can be processed in two phases: by a self-organized training phase and a recognition phase. The lowest stage of the network is the input layer U_0 . Each of the succeeding i -th stages has a layer U_{S_i} consisting of S-cells followed by a layer U_{C_i} of C-cells. Each layer of S-cells or C-cells is composed by a number of twodimensional arrays of cells, called *cell-planes*. Each cell-plane is associated to a single feature, extracted from the training patterns during the learning phase. The S-cells are responsible by the feature extration, and the input connection weights of the S-cells are adjusted during the training phase. C-cells are responsible by the distorted features correction. Each cell, or neuron, inside a cell-plane receives input connections from the preceding layer of cell-planes.

To obtain the $u_{S_i}(n, k)$ value, it is firstly computed the weighted sum. $e(n, k)$ of all inputs coming from all K_{l-1} cell-planes of the preceding layer, in a given connection area S_v , which surrounds the position n , of the preceding layer C-cell, or input layer, by the following Eq.(1).

$$e(n, k_l) = \sum_{k_{l-1}=0}^{K_{l-1}} \sum_{i \in S_l} a(i, k_{l-1}, k_l) u_{C_{l-1}}(n_i, K_{l-1}) \quad (1)$$

and the $u_{V_l}(n, k_l)$ by Eq.(2):

$$u_{V_l}(n, k_l) = \sqrt{\sum_{k_{l-1}=1}^{K_{l-1}} \sum_{i \in S_l} c_l(i) \cdot u_{C_{l-1}}^2(k_{l-1}, n + i)} \quad (2)$$

Then the $u_{S_l}(n, k)$ value is obtained by the Eq.(3):

$$u_{S_l}(n, k_l) = \frac{\theta}{1 - \theta} \cdot \varphi \left[\frac{1 + e(n, k_l)}{1 + \theta b(k_l) \cdot u_{V_l}(n, k_l)} - 1 \right] \quad (3)$$

where $\varphi[x] = x$, if $x \geq 0$, and $\varphi[x] = 0$, otherwise.

To obtain the $u_{C_l}(n, k)$ value, it is at first computed the weighted sum of all inputs corresponding to the previously obtained $u_{S_l}(n, k)$, in a given connection area S_v , which surrounds the position n , of the preceding S-cell layer, by following Eq.(4):

$$u_{C_l}(n, k) = \psi \left[\sum_{i \in S_l} d_l(i) \cdot u_{S_l}(n + i, k_l) \right] \quad (4)$$

where $\psi[x] = \frac{x}{1+x}$ if $x \geq 0$, and $\psi[x] = 0$, otherwise.

Fig. 1 shows a tree-stage neocognitron structure of S-cells and C-cells. All k cell planes from stage $l = 1$ take the input values from the input layer. After the parallel processing of the k cell planes in layer l , all results are used by all cell planes of the layer $l + 1$, as shown by the interconnections between layers. Data are spread into all cell planes,

which occur after the layer processing. The last stage is connected to the output layer, which corresponds the network result layer.

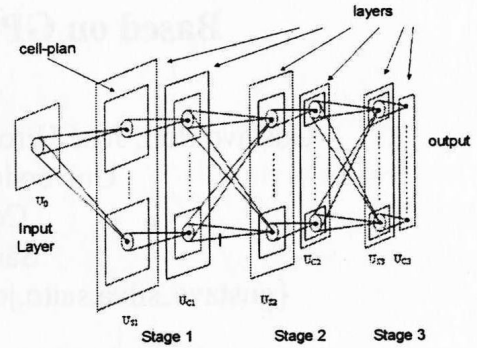


Figure 1. Neocognitron structure with three stages of S-Cells (u_{S_l}) and C-cells(u_{C_l})

3 Compute Unified Device Architecture

Compute Unified Device Architecture (CUDA) [5] is a new hardware and software architecture for issuing and managing computations on the GPU as a data-parallel computing device without need of mapping them to a graphics API. When programmed through CUDA, the GPU is viewed as a compute device of executing a very high number of threads in parallel. It operates as a coprocessor to the main CPU, or host.

Each multiprocessor, on computer device (GPU), is implemented as Single Instruction, Multiple Data architecture (SIMD), so that at any given clock cycle, each processor of the multiprocessor executes the same instruction, but operates on different data.

The CUDA software stack is composed of several layers as illustrated in Fig. 2, a hardware driver, an application programming interface (API) and its runtime, and two higher-level mathematical libraries of common usage, Fast Fourier Transform 1D, 2D and 3D transforms of complex-valued signal data (CUFFT) and an implementation of basic linear algebra (CUBLAS). The hardware has been designed to support lightweight driver and runtime layers, resulting in high performance.

The batch of threads that executes a kernel is organized as a grid of thread blocks in a batch of threads that can cooperate together sharing data through some fast shared memory and synchronizing their execution to coordinate memory accesses efficiently. Each thread is identified by its thread ID, which is the thread number within the block and it specify a block as a 2 or 3 dimensional array of arbitrary size. For a 2 dimensional block size (D_x, D_y) the thread ID of a thread of index (x, y) is $(x + yD_x)$ and for

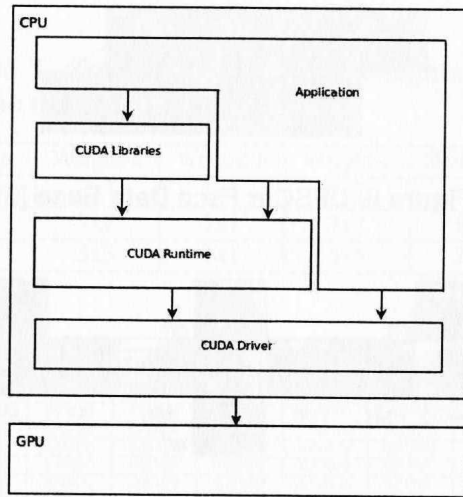


Figure 2. CUDA Stack

a 3 dimensional block size (D_x, D_y, D_z) the thread ID of a thread of index (x, y, z) is $(x + yD_x + zD_xD_y)$ (Fig. 3).

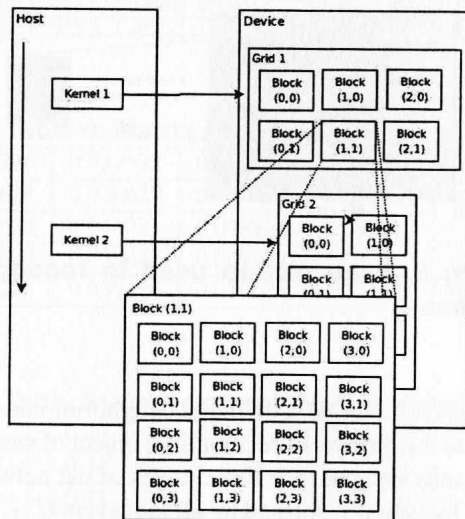


Figure 3. Thread Batching

A grid of thread blocks is executed on device by executing one or more block on each multiprocessor using time slicing. Each block is split into SIMD groups of threads called warps, each of these warps contains the same number of threads, called the warp size, and is executed by the multiprocessor in a SIMD fashion; a thread scheduler periodically switches from one warp to another to maximize the use of the multiprocessor's computational resources.

Device memory (Fig. 4) can be allocated either as linear memory or as CUDA array and a thread that executes on the device has only access to the device's DRAM and on-chip memory through the following memory spaces:

- Read-write per-thread registers;

- Read-write per-thread local memory;
- Read-write per-block shared memory;
- Read-write per-grid global memory;
- Read-only per-grid constant memory; and
- Read-only per-grid texture memory.

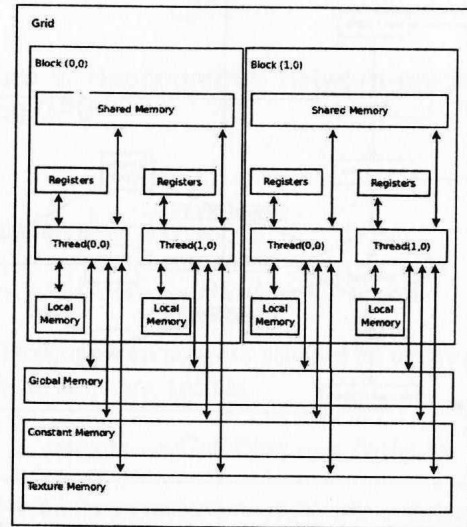


Figure 4. Memory Model

Texture memory also offers different addressing modes, as well as data filtering, for some specific data formats.

The CUDA API comprises an extension to the C programming and for the compilation of the CUDA source files it is used the NVCC, a compiler driver that simplifies the process of compiling. It provides simple and familiar command line options and executes them by invoking the collection of tools that implement the different compilation stages. NVCC's basic workflow consists in separating the device code from the host code and compiling the device code into a binary form or cubin. The generated host code is output either as C code that is left to be compiled using another tool or as object code directly by invoking the host compiler during the last compilation stage.

4 Methodology

A face recognition system using neocognitron neural network can be processed in two phases: the learning phase, and the recognition phase. This work have focus on recognition, since, the learning phase is ready. The training is being carried out by an application developed in Delphi [8] and it has as a product of its activity 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 C-cells of the stage.

At the recognition phase, one face image (input pattern) is shown to the system, and it process and identifies the face. Fig. 5 shows the block diagram of the parallel processing management algorithm of the face recognition phase using CUDA, in two parts, host an device.

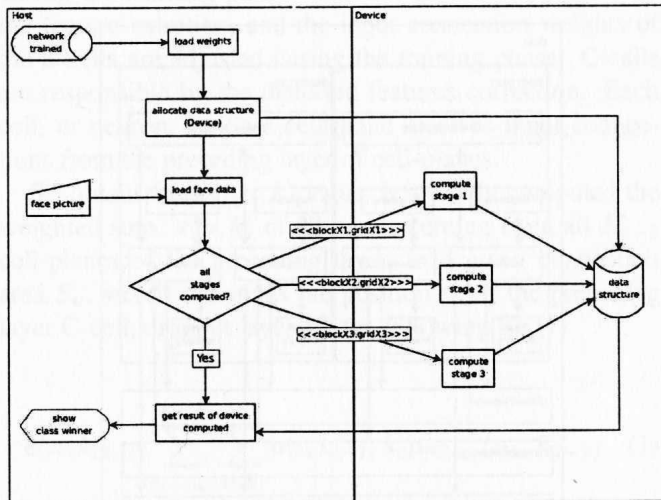


Figure 5. Block Diagram Organization of this Work

As can be seen in the block diagram (Fig. 5), 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 by UFSCar [3] (Fig.6) and the CMU PIE database [10] (Fig.7) 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° . Other 4 cameras include 2 above and below the central camera, and 2 in the corners of the room. On Fig. 7, the different positions of cameras are identified by ratings c02m ... 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.

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) [10] 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



Figure 6. UFSCar Face Data Base [3]

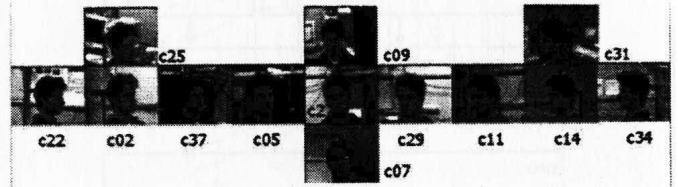


Figure 7. CMU PIE database [10]

capture of the face region, the reduced image of size 57x57, as shown in Fig. 8.

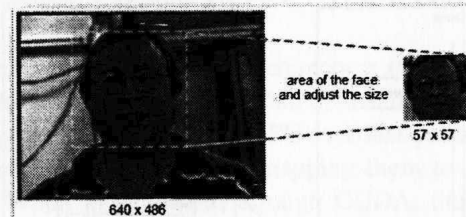


Figure 8. Face picture used to recognition process

At Fig. 9, it can be seen the neocognitron network processing, to the process of recognition, object of study of this work. 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 1 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, in another work of the authors[8].

The *weight-c* and *weight-d* are fixed and defined at the time of implementing the network. On the Table 2, 3 and 4 are presented as matrices *weight-c* used in stages 1, 2 and 3 respectively.

Tables 5, 6 and 7 corresponds to the matrices of *weight-d* used in the processing of stages 1, 2, and 3, respectively.

To carry out processing of any type of data within the CUDA, it must be able to be implemented within a hierar-

Table 1. dimensionality of the weights used in the network

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 2. 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 3. Matrix of weight-c of stage 2

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 4. Matrix of weight-c of stage 3

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 5. Matrix of weight-d of stage 1

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 6. Matrix of weight-d of stage 2

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 7. Matrix of weight-d of stage 3

0.81	0.9	0.81
0.9	1	0.9
0.81	0.9	0.81

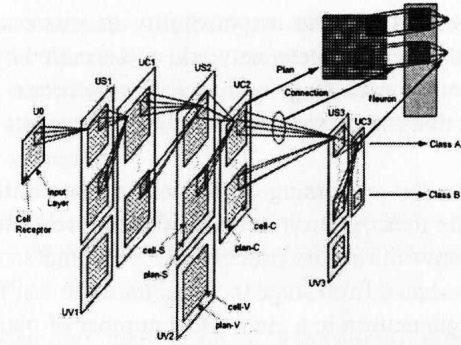


Figure 9. Neocognitron network processing using GPU

chical organization such as:

Grid $\mapsto$ *Block* $\mapsto$ *Thread*

The Neocognitron network also has an organization in a hierarchical structure, such as:

Stage $\mapsto$ *CellPlan* $\mapsto$ *Neuron*

Analyzing the organization of the two architectures, it is possible to verify some points of similarity, which can be seen in Fig. 9 and listed in Table 8, 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 equivalence between the two architectures facilitates the network modeling to be used at the CUDA/GPU environment.

Table 8. Points of Similarity: CUDA x Neocognitron

CUDA	Neocognitron
Grid	Stage
Block	Cell Plan
Thread	Neuron

Another important factor of the validity of the equivalence of the architectures lies in the fact that there is independence of the values of neurons at their processing. That is, the value of a neuron, a cellular plan, does not depend on the value of the neighbor neuron at the same plane, but on the huge data of the preceding stage. That validates the use of an 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 Fig. 5.

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 modeled 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, that 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 call are shown in Table 9. For each stage took place in consideration the size of the area of connection. The goal is to process a plan with the greatest number of areas of possible connections.

Table 9. Organization of cell-plans and connection area and its implementation in GPU/CUDA

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

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 area of connection area, of which there are 5 blocks in the grid processing simultaneously, or 245 connections are computed simultaneously, as can be seen in Fig. 10.

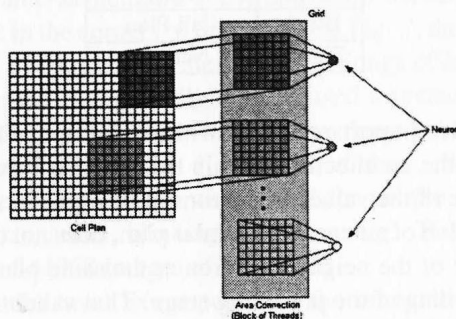


Figure 10. Neuron connection processing diagram

The same analysis applies to all stages of the network

shown in Table 9, and the total number of neurons processed simultaneously from 245, 245 and 250, to the stages 1, 2 and 3 respectively.

5 Results

Using the UFSCar [3] 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 10.

Table 10. Degree of accuracy of recognition of faces

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

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.

Table 11. Time, in seconds, spent for processing by cell-plan

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

Table 12 presents a comparison between the processing time in the GPU / CUDA, with the same network being processed in a single environment and processed in a cluster with 8 processors values obtained by Ribeiro [7]. It was made an adjustment in time goals for the work Ribeiro [7] 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.

By this table it is possible to calculate the speed-up and efficiency of processing between the architectures. These values are presented in Table 13, organized into three columns: computer architecture, Speed-up, and efficiency.

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 14, in two columns, the first "Reserve Area" indicates

Table 12. Comparing the processing time in different architectures

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

Table 13. Comparison Speed-up and Efficiency in different architectures

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

where it allocated the amount of memory in Mega Bytes presented in the second column "Located Area".

Table 14. Number of dedicated memory, of GPU, allocated for the implementation

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

6 Conclusion

With the processing of Neocognitron network 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 network Neocognitron 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 probably 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 environments, 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 modeling of streaming data processing. As seen in other studies [6], 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, partly 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.

7 Acknowledgments

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.

References

- [1] K. Fukushima and S. Miyake. Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shift in position. *Pattern Recognition*, 15(6):455-469, 1982.
- [2] K. Fukushima and N. Wake. Improved neocognitron with bend-detection cells. *IEEE - International Joint Conference on Neural Networks*, 1992.
- [3] M. H. Hirakuri. Aplicação de rede neural neocognitron para reconhecimento de atributos faciais. *Dissertação de Mestrado - Universidade Federal de São Carlos*, 2003.

- [4] T. Noticias. China implanta sistemas de reconhecimento facial biométrico. <http://noticias.terra.com.br/ciencia/interna/0,,OI956783-EI238,00.html>, 2006.
- [5] NVIDIA. *NVIDIA CUDA Compute Unified Device Architecture - Programming Guide*. NVIDIA, 2007.
- [6] G. Poli, A. M. L. Levada, J. F. Mari, and J. H. Saito. 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*, pages 19–27, 2007.
- [7] L. J. Ribeiro. Paralelização da rede neural neocognitron em cluster smps. *Dissertação de Mestrado da Universidade Federal de São Carlos (UFSCar)*, 2002.
- [8] J. H. Saito and S. Abib. Using cmu pie human face database to a convolutional neural network - neocognitron. *ESANN2005 - European Symposium on Artificial Neural Network*, pages 491–496, 2005.
- [9] J. H. Saito and K. Fukushima. Modular structure of neocognitron to pattern recognition. *ICONIP'98, Fifth Int. Conf. on Neural Information Processing, Kitakyshu - Japan*, 1998.
- [10] T. Sim and M. Bsat. The cmu pose, illumination, and expression database. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, pages 1615–1618, 2003.