Artificial neural network computation on graphic process unit

Conference Paper · January 2005

DOI: 10.1109/IJCNN.2005.1555903 · Source: IEEE Xplore

Zhongwen Luo
China University of Geosciences
22 PUBLICATIONS **230** CITATIONS
SEE PROFILE

Hongzhi Liu
Peking University
52 PUBLICATIONS **409** CITATIONS
SEE PROFILE

Artificial Neural Network Computation on Graphic Process Unit

Zhongwen Luo, Hongzhi Liu and Xincai Wu

Faculty of Information, China University of Geoscience(Wuhan) , Wuhan 430074,China
E-mail : luozw@cug.edu.cn,liuhz87.student@sina.com, xcwu@cug.edu.cn

Abstract

Artificial Neural Network (ANN) is widely used in pattern recognition related area. In some case, the computational load is very heavy, in other case, real time process is required. So there is a need to apply a parallel algorithm on it, and usually the computation for ANN is inherently parallel. In this paper, graphic hardware is used to speed up the computation of ANN. In recent years, graphic processing unit (GPU) grows faster than CPU. Graphic hardware venders provide programmability on GPU. In this paper, application of commodity available GPU for two kinds of ANN models was explored. One is the self-organizing maps (SOM); the other is multi layer perceptron (MLP). The computation result shows that ANN computing on GPU is much faster than on standard CPU when the neural network is large. And some design rules for improve the efficiency on GPU are given.

Keywords: Graphic Process Unit; ANN; SOM; MLP

I. INTRODUCTION

Artificial neural network is widely used in classification and pattern recognition. In this paper, two kinds of ANN, self-organizing maps (SOM) [1] and multi layer perceptron (MLP), are implemented on graphic hardware for speed up the computation.

MLP is a very simple neural network, the process is linear with one input layer, several hidden layer, and an output layer. It is usually trained by back propagation (BP) algorithm. SOM consists of one layer of n-dimensional units (neurons). It is fully connected with the network input. Additionally, there exist lateral connections through which a topological structure is imposed. For the standard model, the topology is a regular two-dimensional map instantiated by connections between each unit and its direct neighbors.

For relative works, Kyoung-Su Oh et al.[2] implement an GPU based MLP for classify the text area in a image, and give an almost 20 time speed up over CPU. Thomas Rolfes[3] gives an artificial neural network implementation using a GPU-based BLAS level 3 style single-precisions general matrix-matrix product. Bohn[4] describes an SOM calculation method based on OpenGL hardware speed-up on SGI workstation, which inspired our work to further deploy the possibility to implement ANN calculation based on PC commodity graphic hardware.

In recent years, the graphic hardware performance is doubled every 12 months which is much faster than CPU's performance increase speed which is doubled every 18 months. And GPU vendors had make programmability on GPU, which make it possible for implement general-purpose computation.

In this paper, two kinds of ANN computation on GPU are given. In section 2, SOM computation model and implementation on graphic hardware is discussed. In section 3, MLP computation model and implementation on GPU is discussed. In section 4, the computation result and comparison are given for both CPU and GPU. In section 5, some of the design details and lessons we learned during implementation are given. In section 6, conclusion and some future works are given.

II. COMPUTATIONAL MODEL AND IMPLEMENT METHOD FOR SOM

A. The SOM Computational Model

The SOM takes a two-step computation: search for the best matching unit (BMU) and modify the map according to a distance function of the lateral connections. Usually, the Euclid distance is chosen as similarity measure method. The calculation formula takes as follow:

$$\| W_b - \xi \| < \| W_i - \xi \| \quad \text{for any i;} \qquad [1]$$

Modify the unit value regarding a distance function of the lateral connections as follow.

$$W_i^{new}=W_i^{old}-\varepsilon\Omega(r_b,r_i)*(W_i^{old}-\xi) \qquad [2]$$

As Bohn described, three OpenGL[5] extension functions are needed, they are blending, glColorMatrix and glminmax. These functions are fully support by SGI workstation, but only partially supported by PC graphic hardware. So it is difficult to fully implement SOM on OpenGL. Fortunately, current GPU provide more programmability, which makes it possible for implement the SOM calculation on GPU.

B. SOM Implementation on Programmable GPU

As discussed in the previous part, not all OpenGL functions supported by SGI workstation are supported by PC commodity graphic hardware. But recently, graphic hardware vendors provide programmability and some high-level program language[6,7]. This kind of programmability is at a lower level than OpenGL, and is more powerful than fixed OpenGL function. In this implementation, Cg [8] (C for graphic) is chosen as developing environment.

As described in previous part, the calculation contains two steps. The first is finding the best matching unit, and the second is adjusting the value according to the distance from BMU.

For finding the BMU, two steps are needed. In the first step, similarity measurement is calculated; in the second step, the minimum values which represent the BMU are found and located. The similarity computation code for GPU takes as follow:

```
Half 4 temp= tex2D(texture,coords) -intrain;

c.x = dot(temp,temp);

c.yz= coords.xy;
```

The first two sentences calculate the square of Euclid distance of two vectors, the third sentence save the unit coordinate, which will be used late for locating the best match unit.

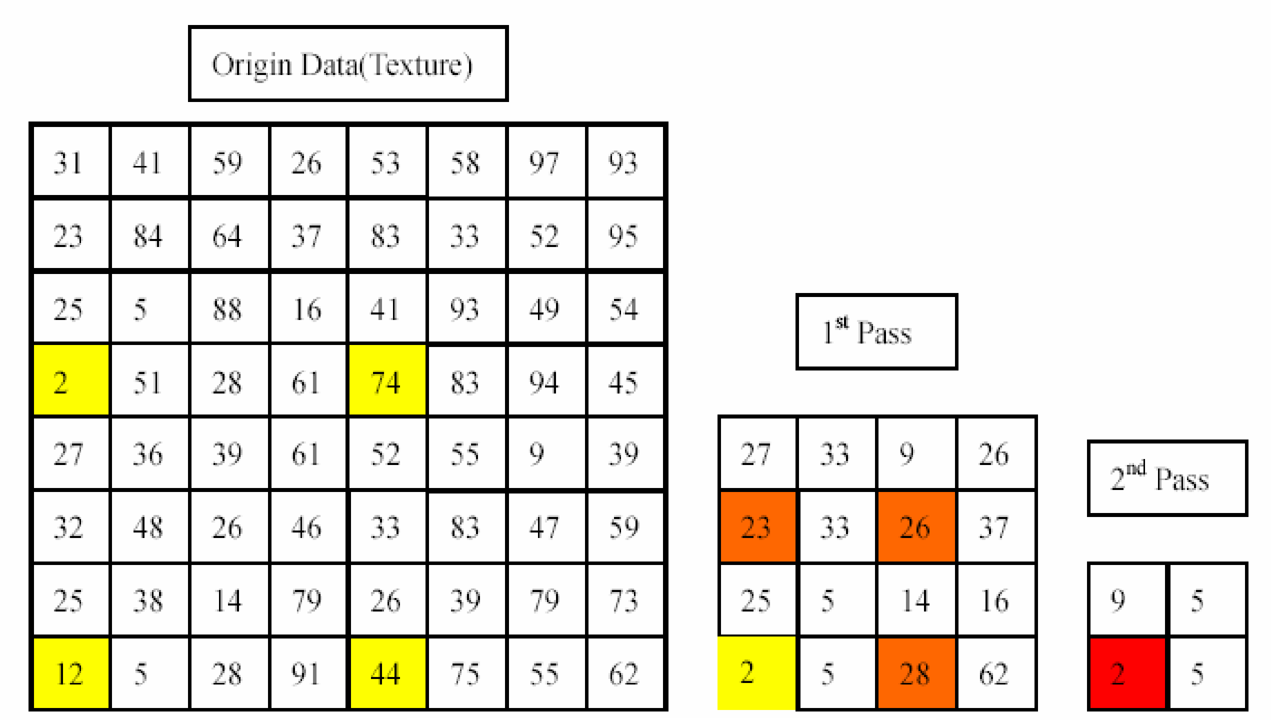
The main difficulty in Cg computation comes from finding the minimum value and determination its location, for there is no global variable in Cg environment. We use a multipass method to calculate it. Our scheme shows as Fig 1. In each pass we find the minimum value of four units show as colored and save the result in a small size texture. After some step, the size decrease to 1, and we can get the minimum value and its location.

After get BMU, we can adjust the self-organize map according to equation [2].

III. MLP FOR REAL TIME BALL RECOGNIZING IN FIRA SOCCER ROBOT

A. Background

Currently, there are two main world soccer-robot competition, one is RoboCup[9], the other is FIRA[10]. In both case, Object are identified by their unique color. But in the near future, these color cues may be removed, so new vision algorithms based on model are needed to cope with the situation. The model based algorithm for finding ball



Origin Data(Texture)

31	41	59	26	53	58	97	93
23	84	64	37	83	33	52	95
25	5	88	16	41	93	49	54
2	51	28	61	74	83	94	45
27	36	39	61	52	55	9	39
32	48	26	46	33	83	47	59
25	38	14	79	26	39	79	73
12	5	28	91	44	75	55	62

1st Pass

27	33	9	26
23	33	26	37
25	5	14	16
2	5	28	62

2nd Pass

9	5
2	5

Fig. 1: Scheme for find the minimum value

and non-ball location will discussed below.

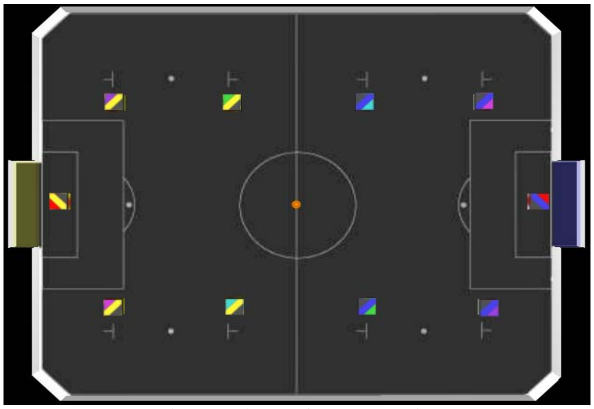


Fig 2. An image from FIRA SimuroSot

Fig2 shows an image taken from FIRA robot soccer 5vs5 simulator. Our task is to develop a model to recognize the ball, and then using this model to recognize the ball in real time.

For each location, the characters are calculated considering the color value at a small area around the location. The area radius takes as 7, which is shown in Fig 3,

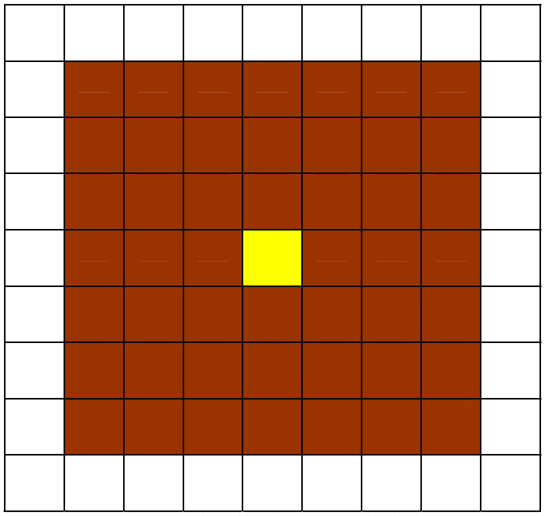


Fig 3. region for character calculation

Seven parameters are selected as characters of the ball. There are 3 average value of the colors red, green and blue around the concerning position:

$$mean_{x,y} = \sum_{i=-3}^{3}\sum_{j=-3}^{3} c_{x+i,y+j} \Big/ 49$$

3 standard deviation of the colors red green and blue:

$$delta_{x,y} = \sqrt{\sum_{i=-3}^{3}\sum_{j=-3}^{3}(c_{x+i,y+j} - mean_{x,y})^2 \Big/ 48}$$

And the luminance:

$$luminance = r + g + b$$

B.Computational Model of MLP

Three layer MLP neural network is selected to recognize the ball. The input layer consists of seven nodes for seven characters. The hidden layer consists of three nodes, and the output layer consists of just one node. Back propagation method was chosen to train the network.

The train set takes as follow. 16 locations are selected from each of 10 robot cars, 4 locations are selected from center of ball, and one location is selected from background field. Totally, 165 locations are selected as the train set. For each selected location, seven characters are calculated and take as an element of the train set.

The trained MLP is used to recognize and trace the ball in real time.

There are two main computation steps for MLP used as the classification machine. The first one is matrix multiplication:

$$net = w \bullet x + b \qquad [3]$$

The second one is sigmoid function calculation:

$$\sigma(net) = \frac{1}{1+e^{-net}} \qquad [4]$$

As for the determination of ball on robot soccer, the MLP calculation was applied on each point in the play ground. And MLP was used to distinguish between ball and non-ball position.

c. MLP Implementation on Graphic Hardware

Current GPU has a limited instruction length and a limited number of temporary variables for calculation at each location. So multipass is needed for complex problem. For MLP discussed in this section, three passes are performed. In the first pass, average values of three color and luminance are calculated. In the second pass, the standard deviations of the three colors are calculated. In the third pass, classification result is got from MLP calculation on the characteristic.

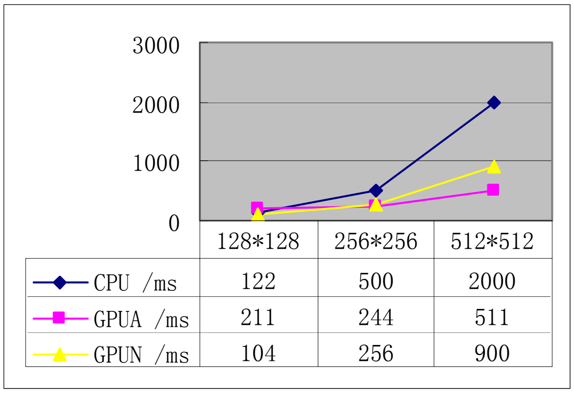
In this calculation, we use a new Nvidia's GF6000 serious graphic hardware. The reason is that the ATI's GPU supports very few numbers of operations in each pass; so more passes are need for MLP calculation. Old Nvidia's GPU does not support fully float texture, which is crucial for the precision of result.

IV. RESULTS AND DISCUSSION

The environment for CPU computing is INTEL P4 2.4G. Based on this PC, ATI 9550 and Nvidia GF5700 GPU are used for SOM computing, Nvidia GF6200 GPU is used for MLP computing.

A. CPU and GPU Train Time for SOM and Discussion

Usually the train process for self organize map is time consuming when the map is large enough. So there is a need to speed up the training procedure. For our test problem, 80 data are chosen for training the SOM and average time for the computation on CPU and GPU can be reached. The result is shown in Fig 4.



	128*128	256*256	512*512
CPU /ms	122	500	2000
GPUA /ms	211	244	511
GPUN /ms	104	256	900

Fig. 4: SOM train computing time on GPU and CPU

The result shows that GPU based implementation is faster than CPU, especially for large self organize map. As map size increase, the computation time on GPU increase slowly than that on CPU. And different GPU had different result, for Nvidia's GPU, it takes the least time for small size SOM like 128*128, but for ATI's GPU it takes the least time for larger size SOM like 256*256. The difference may come from vendor's hardware implementation. The result shows that ATI's graphic card takes more time for the compile of program and the code is better optimized so computation time decrease with more data, but Nvidia's graphic card takes less time for compile and takes more time to computing.

B. MLP Computation Time for CPU and GPU

The application of MLP in this paper is to trace the ball in robot soccer in real time. The result is shown in table 1.

The result shows that GPU based MLP computation is about 200 times faster than that of CPU. And the result also shows that GPU computation is fast enough for the locating of the ball in real time.

TABLE I: MLP COMPUTATION ON CPU AND GPU.

CPU /ms	11328
GPUN /ms	46

V. SOME IMPLEMENTATION DETAILS AND LESSONS

To increase the efficiency, some basic rule should obey. The main rules are given below.

First, create the GPU hardware program only once and enable it when it is needed. The reason is that when a new program is created, it will be compiled by Cg, which is time consuming.

Second, if possible, do one's best to decrease the calculation passes. For computation scheme described in section 2.2, the main bottleneck is at finding minimum procedure. The test shows that the bottleneck comes from multi-pass used in finding minimum procedure. Table 2 shows result for different scheme of finding the minimum. To decrease the pass, we make two changes, the first is to combine the value calculation with one pass of find minimum, and the second is to combine two pass of find minimum computation into one pass. Instead of calculate 4 units, 16 units are calculated. GPUA represent calculation on ATI graphic card and GPUN represent the calculation on NVIDIA graphic card. The performance increase is clear, especially for Nvidia's graphic card.

TABLE II: COMPARISON BETWEEN MORE AND FEW PASS

KFM size	128*128	256*256	512*512
GPUA more pass /ms	366	400	533
GPUA few pass /ms	211	244	511
GPUN more pass /ms	190	640	2889
GPUN few pass /ms	104	256	900

Third, do best to decrease the data exchange between CPU and GPU. Usually OpenGL's PBuffer are used to save the intermediate result in a texture on GPU and reuse it as an input data. Harris[3] had created a class called "Render to Texture" to easy the use of PBuffer. We had used this class in our program.

Fourth, hardware from different venders usually has different property. So if one implementation is not work at one kind of hardware, try another implementation. For example, in the calculation of minimum value, firstly we use the following code:

```
c=tex2D(texture,coords).x<tex2D(texture,coords+half2(0,offset)).x? tex2D(texture,coords) :
tex2D(texture,coords+half2(0,offset));
```

Which can give the correct result in ATI card, but can not get correct result in an Nvidia's card, we think that the inner parallel schemes makes the difference. To work around it, the above sentence is changed to the following logical equivalent one:

```
half4 c=tex2D(tex,coords);

if (c.x>tex2D(tex,coords+half2(0,half_side)).x)

    c=tex2D(tex,coords+half2(0,half_side));
```

Then the result is correct for both graphic cards.

VI. CONCLUSION

In this paper, implementation for two ANN models on graphic hardware is given. Inherent parallelism of commodity graphic hardware is used to accelerate the computation of ANN. The result shows that GPU is capable for some of ANN calculation, the graphic hardware make it possible for an increasing performance/cost ratio on the area of large size ANN computation.

Compared to Bohn's initial computation on SGI workstation, our implementation has two benefits. One is our calculation is more precise, for we had use the float point computing. The other is that we only use a commodity available graphic card, which is much more easily available than SGI workstation so can be widely used.

The implementation on graphic hardware introduce in this paper has other implicit benefit too. For the SOM computing, a multi-texture or 3-D texture can be used to store the map and make more general SOM computing without the restriction of the vector length of 4. For the MLP used in robot soccer, some graphic hardware have "video in" function, using this kind of graphic hardware; image information can be retrieved directly from camera and store on graphic memory, and don't need to transfer data between CPU and GPU, which will speed the process.

We can also do other general ANN computation on GPU, because GPU provide almost all arithmetic operation, logic operation and some important mathematic function. And for the application of neural network on images, it is more naive to make such computing on a graphic hardware.

For future works, one is to make other kinds of ANN computation on GPU. The other is to further deploy the MLP on more real situation of robot soccer, which include select better parameter and use faster algorithm on GPU.

REFERENCES

[1] Teuvo Kohonen. Self-organizing maps. Springer Verlag, New York, 1997.

[2] Kyoung-Su Oh, Keechul Jung, GPU implementation of neural networks, Pattern Recognition 37, 6, 1311-1314, 2004

[3] Thomas Rolfes. Artificial Neural Networks on Programmable Graphics Hardware, in Game Programming Gems 4,pp373-378, 2004

[4] Bohn, C.A. Kohonen Feature Mapping Through Graphics Hardware. In Proceedings of 3rd Int. Conference on Computational Intelligence and Neurosciences 1998. 1998.

[5] Woo, M., Neider, J., Davis, T., Shreiner, D. OPENGL Programming Guide,Addison-Wisley,1999

[6] Femando R. and Killgard, M.J. The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics Addison-Wisley, 2003

[7] Harris, M. http://www.gpgpu.org/developer/, 2003

[8] Mark, W.R., Glanville, R. S., Akeley, K. and Killgard, M.J. 2003. Cg : A system for programming graphics hardware in a C-like language. ACM Trans. Graph. 22, 3, 896-907

[9] http://www.robocup.com

[10] http://www.fira.net