Self-Organizing Maps Computing on Graphic Process Unit

Luo Zhongwen[1], Liu hongzhi[1], Yang Zhengping[1] Wu Xincai[1*]

1- China University of Geoscience(Wuhan) - Faculty of Information Engineering
Wuhan 430074, China

Abstract. Self-Organizing Maps (SOM) is a widely used artificial neural network (ANN) model. Because of its heavy computation load when the map is big and inherent parallel, there is a need to apply a parallel algorithm on it. As a SIMD parallel processor, Graphic processing unit (GPU) shows a fast growing speed than CPU. And it also provides programmability recently. In this paper, the algorithm and result of SOM computing on GPU has been given. The result shows that GPU can make SOM computing much faster than standard CPU. Some design tricks for improving the efficiency of computing has discussed. Based on the results and current trends in the development of GPU, it is reasonable to expect that graphic hardware will widely used in other ANN computing for getting high-performance.

1 Introduction

The Self-Organizing Maps, also called Kohonen feature map (KFM) [6] is a particular kind of artificial neural network (ANN) model, which 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.

In recent years, the graphic hardware performance is doubled every 12 months which is much faster than CPU's performance increase which is doubled every 18 months. And GPU vendors had made it possible to program on it, which enable us to implement general-purpose computation.

Bohn[1] 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 SOM calculation based on PC commodity graphic hardware. For other related works, Kyoung-Su Oh et al.[4] implements a fast computation of MLP on GPU, and give an almost 20 time speed up over CPU. Thomas Rolfes[7] gives an artificial neural network implementation using a GPU-based BLAS level 3 style single-percision general matrix-matrix product.

In this paper, our approach detail and result are given. In section 2, we give the computational model and deploy the possibility to implement a KFM computing based on PC graphic hardware. In section 3, we give our computation result and compare the performance increase of GPU over CPU and then discuss some of our implementation details. In section 4, we give our conclusion and some directions for further researching.

[*] This work is sponsored by SRF for ROCS, CUG-Wuhan

2 Computational Model and Implement method

2.1 Computational Model and test on Graphic Card

The KFM takes a two step computation: searching for the best matching unit and modifying the value according to a distance function of the lateral connections. For best matching unit search, we usually take a Euclid distance as measure method. The calculation formula takes as follow:

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

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

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

As Bohn described, three OpenGL[8] extension functions were needed, they are blending, glColorMatrix and glminmax. These functions are fully supported by SGI workstation, but only partially supported by PC graphic card. For example, glColorMatrix is not supported by NVIDIA's card and glminmax is not supported by ATI's card. To solve this function "missing" probelm, we can copy the intermediate result to main memory and calculate the result in CPU. The test result is as follow:

CPU /ms	OpenGL Graphic card /ms
120	4000

Table 1: Computational Time for SOM on CPU and OpenGL

It shows that the performance of PC graphic card can not match that of CPU's. The deficiency partially comes from the use of readpixel to copy intermediate result from video memory and partially comes from the use of drawpixel to activate some of the computing.

2.2 CG implementation on PC commodity graphic hardware

As discussed in the previous part of this section, some of the OpenGL vender's extension functions are not supported by PC commodity graphic card. But recently, graphic hardware vendors had provided programmability and some high-level program languages [5] for GPU. In our implementation, we choose Cg [2] (C for graphic) as our develop environment.

As described in section 2.1, the calculation contains two steps. Firstly we need to find the best matching unit (BMU), and then adjust the value according to the distance from BMU.

For the BMU computation, we use a two step computing. In the first step, we calculation similarity as follow:

```
c.x = dot((tex2D(texture,coords)-intrain),(tex2D(texture,coords)-intrain));
c.yz= coords.xy;
```

The first sentence calculates the Euclid distance of two vectors, here the dot function calculate the inner product of two vector, and tex2D function looks at the texture table and return the vector value. The second sentence saves the unit's coordinate.

The main difficulty in CG computation is how to find the minimum value and determine its location, for there is no global variable in CG environment. We use a multi-pass method to calculate it. So the second step for finding the BMU is shown as in Fig 1. In each pass we find the minimum value of four units and save the result in a texture of smaller size. After a few steps, the size decreases to 1, and we can get the minimum value and its location.

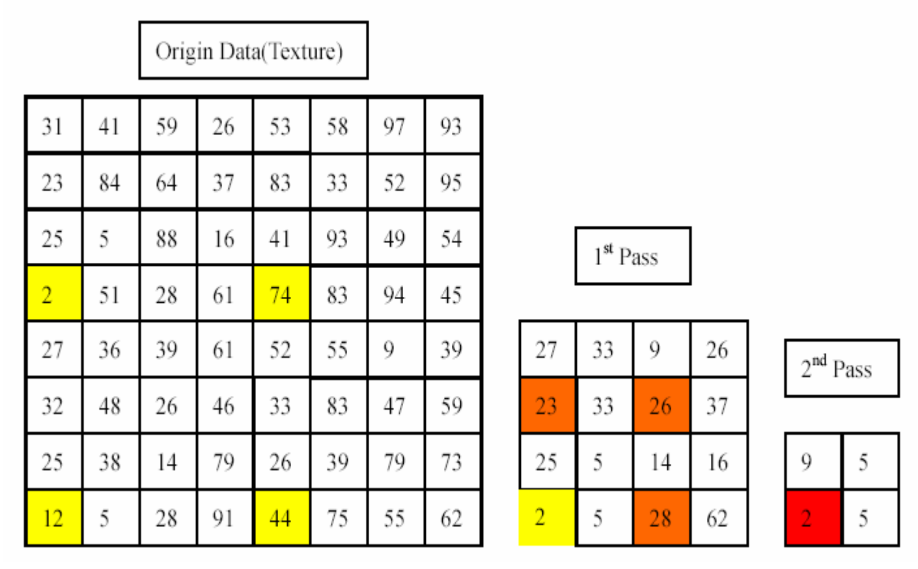


Fig. 1: Scheme for minimum value computing.

After finding BMU, we can adjust the Self-Organize Map according to equation 2.

2.3 Some more implementation detail and lessons

In this section, we will give some implementation tricks and the lessons we learned during our implementation.

Firstly, create same fragment program for only one time. For each time we create a new program, the Cg will compile it, and which is time consuming. We don't notice this at first, and the result is really bad, which makes us almost to give up.

Secondly, if possible, do your best to decrease the number of calculation passes. In our first implementation, we use the scheme described in previous section. We find that the main bottleneck is in finding minimum function. We guess that may because the searching minimum function uses a multi-pass, so we decrease the number of pass by two schemes. First we combine the value calculation with one pass, second in the finding minimum function we calculate 16 units instead of just 4 units that decreases the passes by a factor of two. The result shows in table 2. GPUA is calculated on ATI card and GPUN is calculated on NVIDIA card. We can see the performance increase is clear.

Thirdly, do your best to decrease the data exchange between CPU and GPU. Usually we can use OpenGL's PBuffer to save the intermediate result in a texture and reuse it as a input data. Especially Harris[3] had provided a class Render to Texture to easy the use of PBuffer. We have used this class in our program.

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

Table 2: comparison between more pass and few pass.

3 Result and Discussion

Our test environment is INTEL P4 2.4G for CPU computing. Based on this PC, we had using ATI 9550 and Nvidia 5700 for GPU computing. A typical GPU result shows in Fig 2.

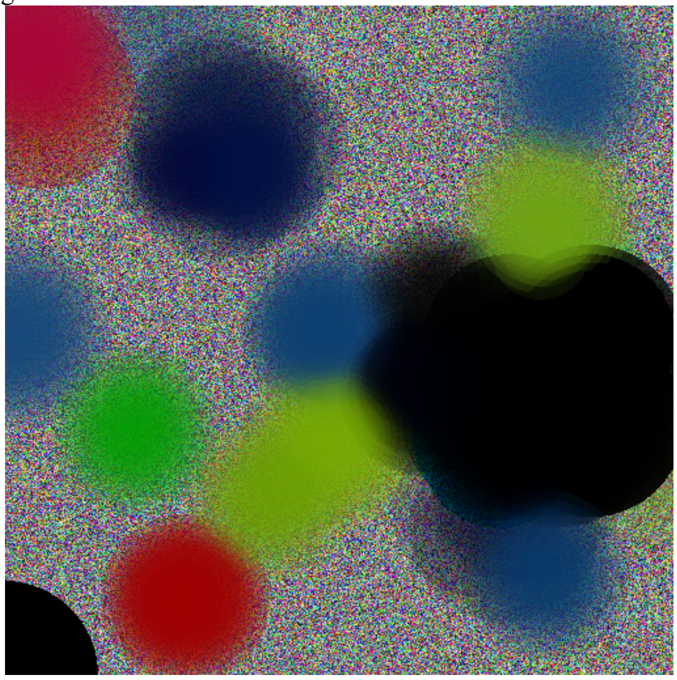
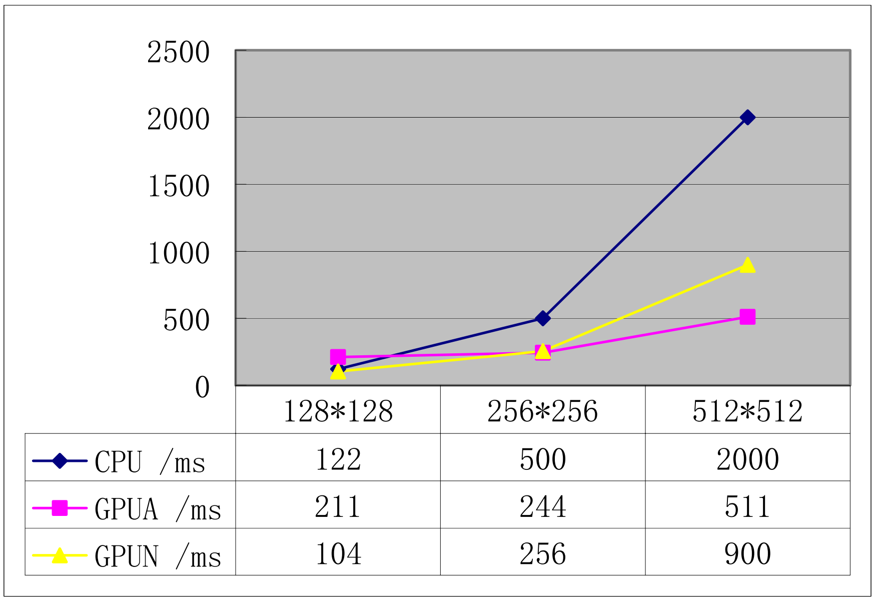


Fig. 2: Typical result for GPU computing.

3.1 CPU and GPU train time and discussion

We had made a series computation for the training of Self-organize Map. We choose 80 data to train the SOM and make an iteration to get the average time for the computation on CPU and GPU. The result shows in Fig 3.

The result shows that our GPU based implementation is faster than CPU, especially for large Self-Organize Map. The increasing of computation time is less than that of CPU. And different GPU had different result, for Nvidia's card, it takes the least time for small size SOM like 128*128, but for ATI's card it takes the least time for larger size SOM like 256*256. We think that the difference comes from vendor's hardware implementation. It seems that ATI's card takes more time for the compiling and loading of program and the code is better optimized so computation time will decrease with more data, and Nvidia's card takes less time for preparing and takes more time to computing.



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

Fig. 3: SOM training computing time on GPU and CPU

3.2 The difference of graphic hardware vender

Apart from the previous difference, there are other differences happened in different graphic hardware venders. For example, for the calculation of minimum value, the following code gives a correct result in ATI card.

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

But the result is wrong in an Nvidia's card. It may because that the inner parallel schemes which makes the difference. To work around it, we change the above sentence to the following logical equivalent one:

```
half4 c=tex2D(texture,coords);
if (c.x>tex2D(texture,coords+half2(0,half_side)).x)
        c=tex2D(texture,coords+half2(0,half_side));
```

And correct result can be obtained in both cards.

4 Conclusion

In this paper, an implementation of Self-Organize Map on graphic process unit has described. We try to use the inherent parallelism of commodity graphic hardware to accelerators the computation of SOM and succeed. The result shows that GPU is capable for the SOM calculation, the graphic hardware make it possible for an increasing performance/cost ratio on the area of high-performance computing.

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 easily available than SGI workstation so can be widely used.

Our implementation on graphic hardware has other implicit benefit too, which is some future works we hope to do. First we can use a multi-texture or 3-D texture to save the map and make more general SOM computing without the restriction of the vector length of 4. We can also do other general ANN computation on GPU, because GPU provides almost all arithmetic operations, logic operations and some important mathematic functions. And for the application of neural network on images, it is more native to make such computing on a graphic hardware.

From above discussion and our test result, we believe as a hardware speedup method, commodity graphic hardware will become increasingly important to high-performance computing, and there will be more results appeared in the neural network computing based on GPU.

References

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

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

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

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

[5] 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

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

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

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